

Lawrence Berkeley National Laboratory

Lawrence Berkeley National Laboratory

Title

FastBit Reference Manual

Permalink

<https://escholarship.org/uc/item/2483c2nj>

Author

Wu, Kesheng

Publication Date

2007-08-02

FastBit Reference Manual

– a0.7 –

Kesheng Wu
Scientific Data Management
Lawrence Berkeley National Lab

August 2007
Report Number: LBNL/PUB-3192

Contents

1 Overview of FastBit IBIS Implementation	1
2 FastBit Namespace Documentation	3
3 FastBit Class Documentation	14
4 FastBit File Documentation	259

1 Overview of FastBit IBIS Implementation

1.1 Introduction

An index in a database system is a data structure that utilizes redundant information about the base data to speed up common searching and retrieval operations. Most commonly used indexes are variants of B-trees, such as B+-tree and B*-tree. FastBit implements a set of alternative indexes call compressed bitmap indexes. Compared with B-tree variants, these indexes provide very efficient searching and retrieval operations by sacrificing the efficiency of updating the indexes after the modification of an individual record.

In addition to the well-known strengths of bitmap indexes, FastBit has a special strength stemming from the bitmap compression scheme used. The compression method is called the Word-Aligned Hybrid (WAH) code. It reduces the bitmap indexes to reasonable sizes and at the same time allows very efficient bitwise logical operations directly on the compressed bitmaps. Compared with the well-known compression methods such as LZ77 and Byte-aligned Bitmap code (BBC), WAH sacrifices some space efficiency for a significant improvement in operational efficiency. Since the bitwise logical operations are the most important operations needed to answer queries, using WAH compression has been shown to answer queries significantly faster than using other compression schemes.

Theoretical analyses showed that WAH compressed bitmap indexes are optimal for one-dimensional range queries. Only the most efficient indexing schemes such as B+-tree and B*-tree have this optimality property. However, bitmap indexes are superior because they can efficiently answer multi-dimensional range queries by combining the answers to one-dimensional queries.

1.2 Key Components

FastBit process queries on one table at a time. Currently, there are two sets of interfaces for query processing, one more abstract and the other more concrete. The more abstract interface is represented by the class `ibis::table` and the more concrete interface is represented by the class `ibis::part`. A table (with rows and columns) is divided into groups of rows called data partitions. Each data partition is stored in a column-wise organization known as vertical projections. At the abstract level, queries on a table produces another table in the spirit of the relational algebra. At the concrete level, the queries on data partitions produce bit vectors representing rows satisfying the user specified query conditions.

1.2.1 Operations on Tables

The main class representing this interface is `ibis::table`. The main query function of this class is `ibis::table::select`, whose functionality resembles a simplified form of the SELECT statement from the SQL language. This function takes two string as arguments, one corresponds to the select clause in SQL and the other corresponds to the where clause. In the following, we will call them the select clause and the where clause and discussion the requirements and restriction on these clauses.

The select clause passed to function `ibis::table::select` can only contain column names separated by comma (.). Aggregate operations such as MIN, MAX, AVG or SUM, are supported through another function named `ibis::table::groupby`. A group-by operation normally specified as one SQL statement needs to be split into

two steps, one to select the values and the other to perform the aggregation operations. We've taken this approach to simplify the implementation. Because these aggregation operations are not directly supported by bitmap indexes, therefore, they are not essential to demonstrate the effectiveness of the bitmap indexes.

The where clause passed to function `ibis::table::select` can be a combination of range conditions connected with logical operators such as AND, OR, XOR, and NOT. Assuming that `temperature` and `pressure` are names of two columns, the following are valid where clauses,

```
temperature > 10000
pressure between 10 and 100
temperature > 10000 and 50 <= pressure and sin(pressure/8000) < sqrt(abs(temperature))
```

The class `ibis::table` also defines a set of functions for computing histograms of various dimensions, `ibis::table`.

Using FastBit, one can only append new records to a table. These operations for extending a table is defined in the class `ibis::tablex`.

For most fixed-sized data, such as integers and floating-point values, FastBit functions expects raw binary data and also store them as raw binary, therefore the data files and index files are not portable across different platforms. This is common to both `ibis::table` interface and `ibis::part` interface. However, one difference is that `ibis::table` handles string values as `std::vector<std::string>`, while the lower level interface `ibis::part` handles strings as raw `char*` with null terminators.

1.2.2 Operations on Data Partitions

The two key classes for query processing on a data partition are `ibis::part` and `ibis::query`, where the first represents the user data (or base data) and the second represents a user query. An `ibis::part` is primarily a container of `ibis::column` objects and some common information about the columns in a data partition. The class `ibis::column` has two specialization for handling string values, `ibis::category` for categorical values and `ibis::text` for arbitrary text strings.

The user query is represented as an `ibis::query` object. Each query is associated with one `ibis::part` object. The functions of the query class can be divided into three groups, (1) specifying a query, (2) evaluating a query, and (3) retrieving information about the hits. The queries accepted by FastBit are a subset of the SQL SELECT statement. Each query may have a WHERE clause and optionally a SELECT clause. Note that the FROM clause is implicit in the association with an `ibis::part`. The WHERE clause is a set of range conditions joined together with logical operators, e.g., "`A = 5 AND (B between 6.5 and 8.2 OR C > sqrt(5*D))`." The SELECT clause can contain a list of column names and some of the four functions AVG, MIN, MAX and SUM. Each of the four functions can only take a column name as its argument. If a SELECT clause is omitted, it is assumed to be "SELECT count(*)". We refer to this type of queries as count queries since their primary purpose is to count the number of hits.

To evaluate a query, one calls either `ibis::query::estimate` or `ibis::query::evaluate`. After a query is evaluated, one may call various function to find the number of hits (`ibis::query::getNumHits`), the values of selected rows (`ibis::query::getQualifiedInts`, `ibis::query::getQualifiedFloats`, and `ibis::query::getQualifiedDoubles`), or the bitvector that represents the hits (`ibis::query::getHitVector`).

1.2.3 Indexes

The indexes are considered auxiliary data, therefore even though they involve much more source files than `ibis::part` and `ibis::query`, they are not essential from a users point of view. In FastBit, the indexes are usually built automatically as needed. However, there are functions to explicitly force FastBit to build them through `ibis::table::buildIndex`, `ibis::part::buildIndex` and their variants.

Currently, all indexes are in a single class hierarchy with `ibis::index` as the abstract base class. The most convenient way to create an index is calling the class function `ibis::index::create`. One can control what type of bitmap index to use by either specifying an index specification for a whole table by calling `ibis::table::indexSpec`, for a whole data partition by calling `ibis::part::indexSpec`, or for each individual column by calling `ibis::column::indexSpec`. The index specification along with other metadata are written to a file named

-part.txt in the directory containing the base data and the index files. The directory name is needed when constructing an `ibis::part`. This information may be indirectly provided through an RC file specified to the function `ibis::init`.

1.3 Acknowledgments

The author gratefully acknowledges the support from Kurt Stockinger, Ekow Otoo and Arie Shoshani. They are crucial in establishing the foundation of the FastBit system and applying the software to a number of applications. Many thanks to the early users. Their generous feedbacks and suggestions are invaluable to the development of the software.

This work was supported by the Director, Office of Science, Office of Advanced Scientific Computing Research, of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

1.4 Additional Information

Additional information available on the web at <http://sdm.lbl.gov/fastbit> or <http://lbl.gov/~kwu/fastbit>.

2 FastBit Namespace Documentation

2.1 ibis Namespace Reference

The current implementation of FastBit is code named IBIS and most data structures and functions are in the namespace `ibis`.

Classes

- class `ambit`
The multi-level range based (cumulative) index.
- class `bad_alloc`
A specialization of `std::bad_alloc`.
- class `bak`
Maps each value to a lower precision (decimal) values and use the the low precision value as center of the bin.
- class `bak2`
A variation on `ibis::bak`, it splits each bin of `ibis::bak` in two, one for entries less than the mapped value and one for the entries that greater and equal to the mapped value.
- class `bin`
The equality encoded bitmap index with binning.
- class `bitvector`
A data structure to represent a sequence of bits.
- class `bitvector64`
A data structure to represent a sequence of bits.
- class `bord`
Class `ibis::bord` stores all its data in memory.

- class [bundle](#)
The public interface of bundles.
- class [bundle0](#)
The null bundle. It contains only a list of RIDs.
- class [bundle1](#)
The bundle with only one component.
- class [bundles](#)
The bundle with multiple components.
- class [bylt](#)
The two-level range-equality code.
- class [category](#)
A specialized low-cardinality text field.
- class [colDoubles](#)
A class to store double precision floating-point values.
- class [colFloats](#)
A class to store single precision float-point values.
- class [colInts](#)
A class to store integer values.
- class [colLongs](#)
A class to store integer values.
- class [colUInts](#)
A class to store unsigned integer values.
- class [colULongs](#)
A class to store unsigned integer values.
- class [column](#)
The class to represent a column of a data table.
- class [colValues](#)
A pure virtual base class.
- class [compRange](#)
The class [compRange](#) stores computed ranges.
- class [dictionary](#)
Provide a mapping between strings and integers.
- class [direkte](#)
Directly use the integer values as bin number to avoid some intermediate steps.
- class [discretePoisson](#)

Discrete random number with Poisson distribution $\exp(-x/\lambda)$.

- class [discretePoisson1](#)
Specialized version of the Poisson distribution $\exp(-x)$.
- class [discreteZipf](#)
Discrete Zipf distribution: $p(k)$ is proportional to $(v+k)^{-a}$ where $a > 1, k \geq 0$.
- class [discreteZipf1](#)
A specialized case of the Zipf distribution $f(x) = 1/(1+x)$.
- class [discreteZipf2](#)
A specialized version of the Zipf distribution $f(x) = 1/(1+x)^2$.
- class [egale](#)
The multicomponent equality code on bins.
- class [entre](#)
The multicomponent interval code on bins.
- class [fade](#)
The multicomponent range-encoded index.
- class [fileManager](#)
This [fileManager](#) is intended to allow different objects to share the same open file.
- class [fuzz](#)
The two-level interval-equality code.
- class [horometer](#)
Horometer – a primitive timing instrument.
- class [index](#)
The base index class.
- class [keywords](#)
Class `ibis::keywords` defines a boolean term-document matrix.
- struct [lessi](#)
A case-insensitive version of `less` for comparing names of tables, columns, and resources.
- class [mensa](#)
Class `ibis::mensa` contains multiple (horizontal) data partitions (`ibis::part`) to form a logical data table.
- class [MersenneTwister](#)
Mersenne Twister generates uniform random numbers efficiently.
- class [mesa](#)
This class implements the two-side range encoding from Chan and Ioannidis.
- class [meshQuery](#)
The class adds more functionality to `ibis::query` to handle data from meshes.

- class [moins](#)
The multicomponent range code on bins.
- class [nameList](#)
A data structure to store a small set of names.
- class [pack](#)
A two-level index.
- class [pale](#)
A two-level index.
- class [part](#)
The class `ibis::part` represents a partition of a relational table.
- class [qAnyAny](#)
A user specifies this type of query expression with the following syntax,.
- class [qContinuousRange](#)
Simple range condition.
- class [qDiscreteRange](#)
- class [qExpr](#)
The top level query expression object.
- class [qMultiString](#)
- class [qRange](#)
A class to represent simple range conditions.
- class [qString](#)
The class `qString` encapsulates information for comparing string values.
- class [query](#)
A data structure for representing user queries.
- class [range](#)
The range encoded bitmap index based.
- class [rangeJoin](#)
A join is defined by two names and a numerical expression.
- class [relic](#)
The basic bitmap index.
- class [resource](#)
A container for name-value pairs.
- union [rid_t](#)
The object identifiers used to distinguish records.
- class [ridHandler](#)
A class for handling file IO for `ibis::rid_t`.

- class `roster`
A roster list is a list of indices for ordering the values in the ascending order.
- class `sapid`
The multicomponent equality encoded index.
- class `sbiad`
The multicomponent interval encoded index.
- class `selected`
A data structure to store the select clause of a query.
- class `slice`
The bit-sliced index (O'Neil). It used the binary encoding.
- class `tabele`
A trivial class for table with one row and one column.
- class `table`
The abstract table class.
- class `tableList`
A list of tables.
- class `tablex`
The class for expandable tables.
- class `tabula`
A trivial class for table with no columns.
- class `tafel`
An expandable table.
- class `text`
A minimalistic structure for storing arbitrary text fields.
- class `uniformRandomNumber`
A functor to generate uniform random number in the range [0, 1).
- class `zona`
The two-level equality-equality code.
- class `zone`
A two-level index.

Typedefs

- typedef `std::vector< colValues * >` `colList`
- typedef `std::map< const char *, part *, lessi >` `partList`
- typedef `array_t< rid_t >` `RIDSet`

Enumerations

- enum **TYPE_T** {
 UNKNOWN_TYPE = 0, **OID**, **BYTE**, **UBYTE**,
 SHORT, **USHORT**, **INT**, **UINT**,
 LONG, **ULONG**, **FLOAT**, **DOUBLE**,
 CATEGORY, **TEXT** }

Supported data types.

Functions

- **ibis::resource** & **gParameters** ()
The reference to the global configuration parameters.
- void **init** (const int verbose=0, const char *rcfile=0)
Initializes internal resources required by ibis.
- const **ibis::bitvector64** & **outerProduct** (const **ibis::bitvector** &a, const **ibis::bitvector** &b, **ibis::bitvector64** &c)
Compute the outer product of a and b, add the result to c.
- const **ibis::bitvector64** & **outerProductUpper** (const **ibis::bitvector** &a, const **ibis::bitvector** &b, **ibis::bitvector64** &c)
Add the strict upper triangular portion of the outer production between a and b to c.
- **ibis::qExpr** * **parseQuery** (const char *str)
Parse a query string.
- **qExpr** * **parseQuery** (const char *str)
Parse a query string.

Variables

- int **accessIndexInWhole** = 0
- int **gVerbose**
Verbosity level.
- const char * **TYPECODE**
One-character code for the enumeration types.
- const char ** **TYPESTRING**
Human readable version of the enumeration types.

2.1.1 Detailed Description

The current implementation of FastBit is code named IBIS and most data structures and functions are in the name space **ibis**.

The name IBIS could be considered as a short-hand for an implementation of Bitmap Index Searching system or Ibis Bitmap Index System.

2.1.2 Enumeration Type Documentation

2.1.2.1 enum `ibis::TYPE_T`

Supported data types.

Enumerator:

UNKNOWN_TYPE Unknown type, can't really do anything with it.

OID A special eight-byte ID type for internal use.

BYTE One-byte long signed integers.

UBYTE One-byte long unsigned integers.

SHORT Two-byte long signed integers.

USHORT Two-byte long unsigned integers.

INT Four-byte long signed integers.

UINT Four-byte long unsigned integers.

LONG Eight-byte long signed integers.

ULONG Eight-byte long unsigned integers.

FLOAT Four-byte IEEE floating-point numbers.

DOUBLE Eight-byte IEEE floating-point numbers.

CATEGORY Low cardinality null-terminated strings.

TEXT Arbitrary null-terminated strings.

2.1.3 Function Documentation

2.1.3.1 `void ibis::init (const int verbose = 0, const char * rcfile = 0) [inline]`

Initializes internal resources required by ibis.

It must be called by user code before any other functions.

Parameters:

verbose An integer indicating the level of verbosity. A negative number make ibis silent, otherwise the larger it is the more ibis will print out.

rcfile A file containing name-value pairs that specifies parameters for controlling the behavior of ibis. If a file name is not specified, it will attempt to read one of the following file (in the given order).

1. a file named in environment variable IBISRC,
2. a file named `ibis.rc` in the current working directory,
3. a file named `.ibisrc` in the user's home directory.

In an RC file, one parameter occupies a line and the equal sign "=" is required to delimit the name and the value, for example,

```
dataDir = /data/dns
cacheDir = /tmp/ibiscache
```

The minimal recommended parameters of an RC file are

- `dataDir`, which can also be written as `dataDir1` or `indexDir`. It tells ibis where to find the data to be queried. Multiple data directories may be specified by adding prefix to the parameter name, for example, `dns.dataDir` and `random.dataDir`.
- `cacheDir`, which can also be written as `cacheDirectory`. This directory is used by ibis to write internal data for recovery and other purposes.

Note:

If this function is not called, the global variables `ibis::gParameters` must be initialized explicitly before doing anything else. Most useful functions and classes rely on this variable.

2.1.3.2 `const ibis::bitvector64 & ibis::outerProduct (const ibis::bitvector & a, const ibis::bitvector & b, ibis::bitvector64 & c)`

Compute the outer product of `a` and `b`, add the result to `c`.

This should make it possible to use both WAH and BBC compress bitvector classes. It is not likely that we can gain much performance by directly using member variables of `ibit::bitvector` because the unit of compressed data from `ibit::vector` do not fit neatly into `ibis::bitvector64::word_t`.

2.1.3.3 `ibis::qExpr* ibis::parseQuery (const char * str)`

Parse a query string.

The query expression is stored in a binary tree (`ibis::qExpr`). There is a mutual exclusion lock to serialize accesses to this function. The implication is that only one query can be parsed at any given time.

2.1.3.4 `qExpr* ibis::parseQuery (const char * str)`

Parse a query string.

The query expression is stored in a binary tree (`ibis::qExpr`). There is a mutual exclusion lock to serialize accesses to this function. The implication is that only one query can be parsed at any given time.

2.1.4 Variable Documentation

2.1.4.1 `int ibis::gVerbose`

Verbosity level.

The larger the value, the more is printed. Default value is 0. A negative value will disable all printing.

2.2 std Namespace Reference

STL namespace.

Classes

- class **allocator**
STL class.
- class **auto_ptr**
STL class.
- class **bad_alloc**
STL class.
- class **bad_cast**
STL class.
- class **bad_exception**
STL class.
- class **bad_typeid**
STL class.
- class **basic_fstream**

STL class.

- class **basic_ifstream**

STL class.

- class **basic_ios**

STL class.

- class **basic_iostream**

STL class.

- class **basic_istream**

STL class.

- class **basic_istreamstream**

STL class.

- class **basic_ofstream**

STL class.

- class **basic_ostream**

STL class.

- class **basic_ostreamstream**

STL class.

- class **basic_string**

STL class.

- class **basic_stringstream**

STL class.

- class **bitset**

STL class.

- class **complex**

STL class.

- class **deque**

STL class.

- class **domain_error**

STL class.

- class **exception**

STL class.

- class **fstream**

STL class.

- class **ifstream**

STL class.

- class **invalid_argument**
STL class.
- class **ios**
STL class.
- class **ios_base**
STL class.
- class **istream**
STL class.
- class **istreamstream**
STL class.
- class **length_error**
STL class.
- struct **less< char * >**
- struct **less< const char * >**
- struct **less< const ibis::rid_t * >**
- struct **less< ibis::rid_t >**
- class **list**
STL class.
- class **logic_error**
STL class.
- class **map**
STL class.
- class **multimap**
STL class.
- class **multiset**
STL class.
- class **ofstream**
STL class.
- class **ostream**
STL class.
- class **ostreamstream**
STL class.
- class **out_of_range**
STL class.
- class **overflow_error**
STL class.
- class **priority_queue**

STL class.

- class **queue**

STL class.

- class **range_error**

STL class.

- class **runtime_error**

STL class.

- class **set**

STL class.

- class **stack**

STL class.

- class **string**

STL class.

- class **stringstream**

STL class.

- class **underflow_error**

STL class.

- class **valarray**

STL class.

- class **vector**

STL class.

- class **wfstream**

STL class.

- class **wifstream**

STL class.

- class **wios**

STL class.

- class **wistream**

STL class.

- class **wstringstream**

STL class.

- class **wofstream**

STL class.

- class **wostream**

STL class.

- class **wostringstream**
STL class.
- class **wstring**
STL class.
- class **wstringstream**
STL class.

2.2.1 Detailed Description

STL namespace.

3 FastBit Class Documentation

3.1 `array_t< T >` Class Template Reference

Template `array_t` implements a replacement of `std::vector`.

```
#include <array_t.h>
```

Public Types

- typedef const T * **const_iterator**
- typedef T * **iterator**

Public Member Functions

- `array_t` (const char *fn, const off_t begin, const off_t end)
Retrieve a portion of the named file to an array.
- `array_t` (const int fdes, const off_t begin, const off_t end)
Read a portion of an open file into an array.
- `array_t` (ibis::fileManager::storage *rhs, const uint32_t start, const uint32_t nelm)
Construct an array from a section of the raw storage.
- `array_t` (ibis::fileManager::storage &rhs)
Turn a raw storage object into an `array_t` object.
- `array_t` (const `array_t< T >` &rhs, const uint32_t offset, const uint32_t nelm=0)
A shallow copy constructor.
- `array_t` (const `array_t< T >` &rhs)
Shallow copy. Should not throw any exception.
- `array_t` (uint32_t n, const T &val)
Construct an array with n elements of value val.
- `array_t` (uint32_t n)

Construct an array with n elements.

- `array_t ()`

The default constructor. It constructs an empty array.

- `const T & back () const`
- `T & back ()`
- `const T * begin () const`
- `T * begin ()`
- `void bottomk (uint32_t k, array_t< uint32_t > &ind) const`

Return the positions of the k smallest elements.

- `void clear ()`
- `void copy (const array_t< T > &rhs)`

The copy function. It performs a shallow copy.

- `void deepCopy (const array_t< T > &rhs)`

The deep copy function.

- `int empty () const`
- `const T * end () const`
- `T * end ()`
- `iterator erase (iterator i, iterator j)`
- `iterator erase (iterator p)`
- `uint32_t find (const T &val) const`

Return the smallest i such that [i] >= val.

- `uint32_t find (const array_t< uint32_t > &ind, const T &val) const`

Return the smallest i such that [ind[i]] >= val.

- `const T & front () const`
- `T & front ()`
- `void insert (iterator p, const_iterator i, const_iterator j)`
- `void insert (iterator p, uint32_t n, const T &val)`
- `iterator insert (iterator pos, const T &val)`

Insert one value or a list of values before p.

- `void nosharing ()`

Make a not-shared copy of the array if it is actually a shared array.

- `const array_t< T > & operator= (const array_t< T > &rhs)`

Assignment operator. It performs a shallow copy.

- `T & operator[] (uint32_t i)`

Modifiable reference to an element of the array.

- `const T & operator[] (uint32_t i) const`

Non-modifiable reference to an element of the array.

- `void pop_back ()`

Reset the size to zero.

- `void printStatus (std::ostream &out) const`

Print internal pointer addresses.

- void `push_back` (const T &elm)
Add one element.
- uint32_t `read` (const int fdes, const uint32_t begin, const uint32_t end)
Read a portion of an open file.
- void `read` (const char *)
Read an array from the name file.
- void `reserve` (uint32_t n)
Reserve space.
- void `resize` (uint32_t n)
Remove the last element. Resize array.
- uint32_t `size` () const
- void `sort` (array_t< uint32_t > &ind) const
Produce index for ascending order.
- void `stableSort` (array_t< uint32_t > &ind, array_t< T > &sorted) const
A stable sort.
- void `stableSort` (array_t< uint32_t > &ind) const
A stable sort that does not modify the current array.
- void `stableSort` (array_t< T > &tmp)
A stable sort using the provided workspace.
- void `swap` (array_t< T > &rhs)
Exchange the content.
- void `topk` (uint32_t k, array_t< uint32_t > &ind) const
Return the positions of the k largest elements.
- void `write` (FILE *fptr) const
write whole array to an opened file
- void `write` (const char *) const
write whole array to the named file

Static Public Member Functions

- static void `stableSort` (array_t< T > &val, array_t< uint32_t > &ind, array_t< T > &tmp, array_t< uint32_t > &itmp)
This function sorts the content of array val.

3.1.1 Detailed Description

`template<class T> class array_t< T >`

Template `array_t` implements a replacement of `std::vector`.

The main difference is that the underlying memory may be managed by `ibis::fileManager`. In addition, it also implements read and write functions that are not present in `std::vector`.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 `template<class T> array_t< T >::array_t (const array_t< T > & rhs, const uint32_t offset, const uint32_t nelm = 0)`

A shallow copy constructor.

It makes a new array out of a section of an existing array.

3.1.2.2 `template<class T> array_t< T >::array_t (ibis::fileManager::storage & rhs)`

Turn a raw storage object into an `array_t` object.

The input storage object is used by the array. No new storage is allocated.

3.1.2.3 `template<class T> array_t< T >::array_t (ibis::fileManager::storage * rhs, const uint32_t start, const uint32_t nelm)`

Construct an array from a section of the raw storage.

The argument `start` is measured in number of bytes, the second argument `nelm` is the number of element of type `T`.

3.1.2.4 `template<class T> array_t< T >::array_t (const int fdes, const off_t begin, const off_t end)`

Read a portion of an open file into an array.

The argument `fdes` must be a valid file descriptor (as defined in `unistd.h`). It attempt to read `end - begin` bytes from the file starting at offset `begin`.

3.1.2.5 `template<class T> array_t< T >::array_t (const char * fn, const off_t begin, const off_t end)`

Retrieve a portion of the named file to an array.

Prefer memory map if possible.

3.1.3 Member Function Documentation

3.1.3.1 `template<class T> void array_t< T >::bottomk (uint32_t k, array_t< uint32_t > & ind) const`

Return the positions of the `k` smallest elements.

Return the indices of the smallest values in array `ind`.

Note:

The resulting array `ind` may have more than `k` elements if the `k`th smallest value is not a single value. The array `ind` may have less than `k` elements if this array has less than `k` elements.

3.1.3.2 `template<class T> void array_t< T >::deepCopy (const array_t< T > & rhs)`

The deep copy function.

It makes an in-memory version of the array `rhs`.

3.1.3.3 `template<class T> uint32_t array_t< T >::find (const T & val) const`

Return the smallest `i` such that `[i] >= val`.

Assume the array is sorted in ascending order.

3.1.3.4 `template<class T> uint32_t array_t< T >::find (const array_t< uint32_t > & ind, const T & val) const`

Return the smallest `i` such that `[ind[i]] >= val`.

Assuming `ind` was produced by the sort function, it returns the smallest `i` such that `operator[] (ind[i]) >= val`.

3.1.3.5 `template<class T> iterator array_t< T >::insert (iterator pos, const T & val)`

Insert one value or a list of values before `p`.

Return pointer to new elem.

3.1.3.6 `template<class T> void array_t< T >::nosharing ()`

Make a not-shared copy of the array if it is actually a shared array.

This does not guarantee that it would not become shared later. The complete solution is to implement copy-on-write in all functions that modifies an array, but that may decrease performance of this class for rare cases of modifications.

3.1.3.7 `template<class T> T& array_t< T >::operator[] (uint32_t i) [inline]`

Modifiable reference to an element of the array.

Note:

For efficiency reasons, this is not a copy-on-write implementation! The caller has to call the function `nosharing` to make sure the underlying data is not shared with others.

3.1.3.8 `template<class T> void array_t< T >::reserve (uint32_t n)`

Reserve space.

If the current storage object does not have enough space, enlarge the storage object.

3.1.3.9 `template<class T> uint32_t array_t< T >::size () const [inline]`**3.1.3.10** `template<class T> void array_t< T >::stableSort (array_t< T > & val, array_t< uint32_t > & ind, array_t< T > & tmp, array_t< uint32_t > & itmp) [static]`

This function sorts the content of array `val`.

The values of `ind[i]` will be reordered in the same way as the array `val`. The two other arrays are used as temporary storage.

Note:

On input, if array `ind` has the same size as array `val`, the content of array `ind` will be directly used. Otherwise, array `ind` will be initialized to be consecutive integers starting from 0.

If the input array `val` has less than two elements, this function does nothing, i.e., does not change any of the four arguments.

3.1.3.11 `template<class T> void array_t< T >::stableSort (array_t< uint32_t > & ind, array_t< T > & sorted) const`

A stable sort.

It does not change this array, but produces a sorted version in `sorted`.

3.1.3.12 `template<class T> void array_t< T >::stableSort (array_t< uint32_t > & ind) const`

A stable sort that does not modify the current array.

It uses two additional arrays for temporary storage.

3.1.3.13 `template<class T> void array_t< T >::stableSort (array_t< T > & tmp)`

A stable sort using the provided workspace.

The current content is modified to be in ascending order. The argument `tmp` is only used as temporary storage.

3.1.3.14 `template<class T> void array_t< T >::topk (uint32_t k, array_t< uint32_t > & ind) const`

Return the positions of the `k` largest elements.

Return the indices of the in sorted values.

Note:

The resulting array `ind` may have more than `k` elements if the `k`th smallest value is not a single value. The array `ind` may have less than `k` elements if this array has less than `k` elements.

The values are sorted in ascending order, i.e., $[ind[i]] \leq [ind[i+1]]$. This is done so that all sorting routines produce indices in the same ascending order. It should be easy to reverse the order the indices since it only contains the largest values.

The documentation for this class was generated from the following files:

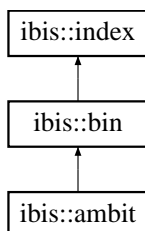
- [array_t.h](#)
- [array_t.cpp](#)

3.2 `ibis::ambit` Class Reference

The multi-level range based (cumulative) index.

```
#include <ibin.h>
```

Inheritance diagram for `ibis::ambit`:



Public Member Functions

- virtual void **adjustLength** (uint32_t nrows)
- **ambit** (const `ibis::bin` &rhs)
- **ambit** (const `ibis::column` *c, `ibis::fileManager::storage` *st, uint32_t offset=8)
- **ambit** (const `ibis::column` *c=0, const char *f=0)
- long **append** (const `ibis::ambit` &tail)
- virtual long **append** (const char *dt, const char *df, uint32_t nnew)

Extend the index.
- virtual void **binBoundaries** (std::vector< double > &) const

The function binBoundaries and binWeights return bin boundaries and counts of each bin respectively.
- virtual void **binWeights** (std::vector< uint32_t > &) const
- virtual void **estimate** (const `ibis::qContinuousRange` &expr, `ibis::bitvector` &lower, `ibis::bitvector` &upper) const

Computes an approximation of hits as a pair of lower and upper bounds.
- virtual long **evaluate** (const `ibis::qContinuousRange` &expr, `ibis::bitvector` &hits) const

To evaluate the exact hits.
- virtual double **getSum** () const

Compute the approximate sum of all the values indexed.
- virtual const char * **name** () const

Returns the name of the index, similar to the function type, but returns a string instead.
- virtual uint32_t **numBins** () const
- virtual void **print** (std::ostream &out) const

Prints human readable information.
- virtual void **read** (`ibis::fileManager::storage` *st)

Reconstructs an index from an array of bytes.
- virtual void **read** (const char *idxfile)

Reconstructs an index from the named file.
- virtual void **speedTest** (std::ostream &out) const

Time some logical operations and print out their speed.
- virtual INDEX_TYPE **type** () const

Returns an index type identifier.
- virtual float **undecidable** (const `ibis::qContinuousRange` &expr, `ibis::bitvector` &iffy) const

Mark the position of the rows that can not be decided with this index.
- virtual void **write** (const char *dt) const

Save index to a file.

Protected Member Functions

- virtual double **computeSum** () const

3.2.1 Detailed Description

The multi-level range based (cumulative) index.

Each level/each bin consists of a range index.

3.2.2 Member Function Documentation

3.2.2.1 `void ibis::ambit::estimate (const ibis::qContinuousRange & expr, ibis::bitvector & lower, ibis::bitvector & upper) const` [virtual]

Computes an approximation of hits as a pair of lower and upper bounds.

Parameters:

expr the query expression to be evaluated.

lower a bitvector marking a subset of the hits. All rows marked with one (1) are definitely hits.

upper a bitvector marking a superset of the hits. All hits are marked with one, but some of the rows marked one may not be hits. If the variable upper is empty, the variable lower is assumed to contain the exact answer.

Reimplemented from [ibis::bin](#).

3.2.2.2 `long ibis::ambit::evaluate (const ibis::qContinuousRange & expr, ibis::bitvector & hits) const` [virtual]

To evaluate the exact hits.

On success, return the number of hits, otherwise a negative value is returned.

Reimplemented from [ibis::bin](#).

3.2.2.3 `double ibis::ambit::getSum () const` [virtual]

Compute the approximate sum of all the values indexed.

If it decides that computing the sum directly from the vertical partition is more efficient, it will return NaN immediately.

Reimplemented from [ibis::bin](#).

3.2.2.4 `void ibis::ambit::print (std::ostream & out) const` [virtual]

Prints human readable information.

Outputs information about the index as text to the specified output stream.

Reimplemented from [ibis::bin](#).

3.2.2.5 `void ibis::ambit::read (ibis::fileManager::storage * st)` [virtual]

Reconstructs an index from an array of bytes.

Intended for internal use only!

Reimplemented from [ibis::bin](#).

3.2.2.6 `void ibis::ambit::read (const char * idxfile)` [virtual]

Reconstructs an index from the named file.

The name can be the directory containing an index file. In this case, the name of the index file must be the name of the column followed by ".idx" suffix.

Reimplemented from [ibis::bin](#).

3.2.2.7 `float ibis::ambit::undecidable (const ibis::qContinuousRange & expr, ibis::bitvector & iffy) const` [virtual]

Mark the position of the rows that can not be decided with this index.

Parameters:

expr the range conditions to be evaluated.

iffy the bitvector marking the positions of rows that can not be decided using the index. Return value is the expected fraction of undecided rows that might satisfy the range conditions.

Reimplemented from [ibis::bin](#).

3.2.2.8 `void ibis::ambit::write (const char * dt) const` [virtual]

Save index to a file.

Outputs the index in a compact binary format to the named file or directory. The index file contains a header that can be identified by the function `isIndex`.

Reimplemented from [ibis::bin](#).

The documentation for this class was generated from the following files:

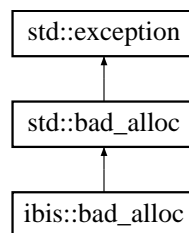
- [ibin.h](#)
- [ixambit.cpp](#)

3.3 `ibis::bad_alloc` Class Reference

A specialization of `std::bad_alloc`.

```
#include <util.h>
```

Inheritance diagram for `ibis::bad_alloc`:



Public Member Functions

- `bad_alloc (const char *m="unknown") throw ()`
- `virtual const char * what () const throw ()`

3.3.1 Detailed Description

A specialization of `std::bad_alloc`.

The documentation for this class was generated from the following file:

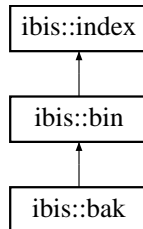
- [util.h](#)

3.4 `ibis::bak` Class Reference

Maps each value to a lower precision (decimal) values and use the the low precision value as center of the bin.

```
#include <ibin.h>
```

Inheritance diagram for `ibis::bak`:



Public Types

- typedef `std::map< double, grain >` **bakMap**

Public Member Functions

- long **append** (const `ibis::bin` &tail)
Append the tail to this index.
- virtual long **append** (const char *dt, const char *df, uint32_t nnew)
Extend the index.
- **bak** (const `ibis::column` *c, `ibis::fileManager::storage` *st, uint32_t offset=8)
- **bak** (const `ibis::column` *c=0, const char *f=0)
- virtual void **binBoundaries** (std::vector< double > &) const
The function binBoundaries and binWeights return bin boundaries and counts of each bin respectively.
- virtual void **binWeights** (std::vector< uint32_t > &) const
- virtual int **contractRange** (`ibis::qContinuousRange` &rng) const
- virtual int **expandRange** (`ibis::qContinuousRange` &rng) const
The functions expandRange and contractRange expands or contracts the boundaries of a range condition so that the new range will have exact answers using the function estimate.
- virtual const char * **name** () const
Returns the name of the index, similar to the function type, but returns a string instead.
- virtual void **print** (std::ostream &out) const
Prints human readable information.
- virtual void **read** (const char *idxfile)
Reconstructs an index from the named file.
- virtual INDEX_TYPE **type** () const
Returns an index type identifier.
- virtual void **write** (const char *dt) const
Save index to a file.

Protected Member Functions

- virtual void **locate** (const [ibis::qContinuousRange](#) &expr, uint32_t &cand0, uint32_t &cand1, uint32_t &hit0, uint32_t &hit1) const
- virtual void **locate** (const [ibis::qContinuousRange](#) &expr, uint32_t &cand0, uint32_t &cand1) const
- virtual uint32_t **locate** (const double &val) const
- void **mapValues** (const char *f, bakMap &bmap) const
- void **printMap** (std::ostream &out, const bakMap &bmap) const

Classes

- struct **grain**

3.4.1 Detailed Description

Maps each value to a lower prevision (decimal) values and use the the low precision value as center of the bin.

It reuses the same variables of [ibis::bin](#), but have to interpret them differently.

Bak is a Dutch word for 'bin'.

3.4.2 Member Function Documentation

3.4.2.1 `int ibis::bak::expandRange (ibis::qContinuousRange & rng) const` [virtual]

The functions `expandRange` and `contractRange` expands or contracts the boundaries of a range condition so that the new range will have exact answers using the function estimate.

The default implementation provided does nothing since this is only meaningful for indices based on bins.

Reimplemented from [ibis::bin](#).

3.4.2.2 `void ibis::bak::print (std::ostream & out) const` [virtual]

Prints human readable information.

Outputs information about the index as text to the specified output stream.

Reimplemented from [ibis::bin](#).

3.4.2.3 `void ibis::bak::read (const char * idxfile)` [virtual]

Reconstructs an index from the named file.

The name can be the directory containing an index file. In this case, the name of the index file must be the name of the column followed by ".idx" suffix.

Reimplemented from [ibis::bin](#).

3.4.2.4 `void ibis::bak::write (const char * dt) const` [virtual]

Save index to a file.

Outputs the index in a compact binary format to the named file or directory. The index file contains a header that can be identified by the function `isIndex`.

Reimplemented from [ibis::bin](#).

The documentation for this class was generated from the following files:

- [ibin.h](#)

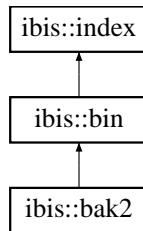
- `idbak.cpp`

3.5 `ibis::bak2` Class Reference

A variation on `ibis::bak`, it splits each bin of `ibis::bak` in two, one for entries less than the mapped value and one for the entries that greater and equal to the mapped value.

```
#include <ibin.h>
```

Inheritance diagram for `ibis::bak2::`



Public Types

- typedef `std::map< double, grain >` **bakMap**

Public Member Functions

- long `append` (const `ibis::bin` &tail)

Append the tail to this index.
- virtual long `append` (const char *dt, const char *df, uint32_t nnew)

Extend the index.
- **bak2** (const `ibis::column` *c, `ibis::fileManager::storage` *st, uint32_t offset=8)
- **bak2** (const `ibis::column` *c=0, const char *f=0)
- virtual void `binBoundaries` (std::vector< double > &) const

The function binBoundaries and binWeights return bin boundaries and counts of each bin respectively.
- virtual void **binWeights** (std::vector< uint32_t > &) const
- virtual int **contractRange** (`ibis::qContinuousRange` &rng) const
- virtual int **expandRange** (`ibis::qContinuousRange` &rng) const

The functions expandRange and contractRange expands or contracts the boundaries of a range condition so that the new range will have exact answers using the function estimate.
- virtual const char * `name` () const

Returns the name of the index, similar to the function type, but returns a string instead.
- virtual void `print` (std::ostream &out) const

Prints human readable information.
- virtual void `read` (const char *idxfile)

Reconstructs an index from the named file.
- virtual INDEX_TYPE `type` () const

Returns an index type identifier.

- virtual void `write` (const char *dt) const
Save index to a file.

Protected Member Functions

- virtual void `locate` (const `ibis::qContinuousRange` &expr, uint32_t &cand0, uint32_t &cand1, uint32_t &hit0, uint32_t &hit1) const
- virtual void `locate` (const `ibis::qContinuousRange` &expr, uint32_t &cand0, uint32_t &cand1) const
- virtual uint32_t `locate` (const double &val) const
- void `mapValues` (const char *f, bakMap &bmap) const
Reads all values and records positions in bmap.
- void `printMap` (std::ostream &out, const bakMap &bmap) const

Classes

- struct `grain`
A simple structure to record the position of the values mapped to the same value.

3.5.1 Detailed Description

A variation on `ibis::bak`, it splits each bin of `ibis::bak` in two, one for entries less than the mapped value and one for the entries that greater and equal to the mapped value.

This way, the index can be used to answer question involving ranges exactly on the mapped values. All internal variables are processed same as a regular `ibis::bin` index.

3.5.2 Member Function Documentation

3.5.2.1 `int ibis::bak2::expandRange (ibis::qContinuousRange &rng) const` [virtual]

The functions `expandRange` and `contractRange` expands or contracts the boundaries of a range condition so that the new range will have exact answers using the function estimate.

The default implementation provided does nothing since this is only meaningful for indices based on bins.

Reimplemented from `ibis::bin`.

3.5.2.2 `void ibis::bak2::print (std::ostream & out) const` [virtual]

Prints human readable information.

Outputs information about the index as text to the specified output stream.

Reimplemented from `ibis::bin`.

3.5.2.3 `void ibis::bak2::read (const char * idxfile)` [virtual]

Reconstructs an index from the named file.

The name can be the directory containing an index file. In this case, the name of the index file must be the name of the column followed by ".idx" suffix.

Reimplemented from `ibis::bin`.

3.5.2.4 `void ibis::bak2::write (const char * dt) const` [virtual]

Save index to a file.

Outputs the index in a compact binary format to the named file or directory. The index file contains a header that can be identified by the function `isIndex`.

Reimplemented from [ibis::bin](#).

The documentation for this class was generated from the following files:

- [ibin.h](#)
- `idbak2.cpp`

3.6 `ibis::bak2::grain` Struct Reference

A simple structure to record the position of the values mapped to the same value.

```
#include <ibin.h>
```

Public Attributes

- `ibis::bitvector * loc0`
- `ibis::bitvector * loc1`
- double `max0`
- double `max1`
- double `min0`
- double `min1`

3.6.1 Detailed Description

A simple structure to record the position of the values mapped to the same value.

The `ibis::bitvector` marked the locations of the values and the min and max record the actual minimum and maximum value encountered.

The documentation for this struct was generated from the following file:

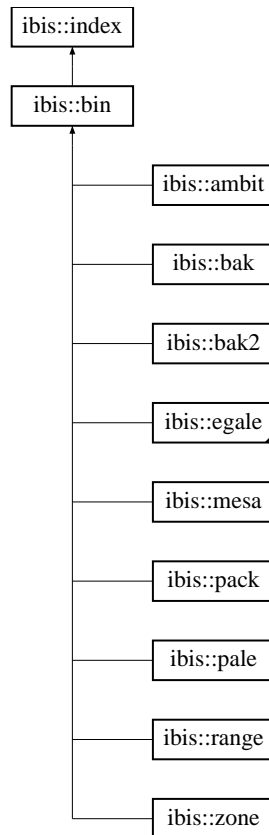
- [ibin.h](#)

3.7 `ibis::bin` Class Reference

The equality encoded bitmap index with binning.

```
#include <ibin.h>
```

Inheritance diagram for `ibis::bin`:



Public Types

- typedef `std::map< double, granule * >` **granuleMap**

Public Member Functions

- long [append](#) (const [array_t](#)< uint32_t > &ind)
Append a list of integers representing bin numbers.
- long [append](#) (const [ibis::bin](#) &tail)
Append the tail to this index.
- virtual long [append](#) (const char *dt, const char *df, uint32_t nnew)
Extend the index.
- **bin** (const [ibis::column](#) *c, const char *f, const `std::vector< double >` &bd)
- **bin** (const [ibis::column](#) *c, const char *f, const [array_t](#)< double > &bd)
- **bin** (const [ibis::column](#) *c, [ibis::fileManager::storage](#) *st, uint32_t offset=8)
- **bin** (const [ibis::column](#) *c=0, const char *f=0)
Construct a bitmap index from current data.
- **bin** (const [ibis::bin](#) &rhs)
- virtual void [binBoundaries](#) (`std::vector< double >` &) const
The function binBoundaries and binWeights return bin boundaries and counts of each bin respectively.
- virtual void **binWeights** (`std::vector< uint32_t >` &) const

- long `checkBin` (const `ibis::qRange` &cmp, uint32_t jbin, const `ibis::bitvector` &mask, `ibis::bitvector` &res) const
Candidate check using the binned values.
- long `checkBin` (const `ibis::qRange` &cmp, uint32_t jbin, `ibis::bitvector` &res) const
Candidate check using the binned values.
- virtual int `contractRange` (`ibis::qContinuousRange` &rng) const
- virtual int64_t `estimate` (const `ibis::bin` &idx2, const `ibis::rangeJoin` &expr, const `ibis::bitvector` &mask, const `ibis::qRange` *const range1, const `ibis::qRange` *const range2) const
- virtual int64_t `estimate` (const `ibis::bin` &idx2, const `ibis::rangeJoin` &expr, const `ibis::bitvector` &mask) const
- virtual int64_t `estimate` (const `ibis::bin` &idx2, const `ibis::rangeJoin` &expr) const
- virtual void `estimate` (const `ibis::bin` &idx2, const `ibis::rangeJoin` &expr, const `ibis::bitvector` &mask, const `ibis::qRange` *const range1, const `ibis::qRange` *const range2, `ibis::bitvector64` &lower, `ibis::bitvector64` &upper) const
- virtual void `estimate` (const `ibis::bin` &idx2, const `ibis::rangeJoin` &expr, const `ibis::bitvector` &mask, `ibis::bitvector64` &lower, `ibis::bitvector64` &upper) const
- virtual void `estimate` (const `ibis::bin` &idx2, const `ibis::rangeJoin` &expr, `ibis::bitvector64` &lower, `ibis::bitvector64` &upper) const
Estimate the number of hits for nonsymmetric joins.
- virtual int64_t `estimate` (const `ibis::rangeJoin` &expr, const `ibis::bitvector` &mask, const `ibis::qRange` *const range1, const `ibis::qRange` *const range2) const
- virtual void `estimate` (const `ibis::rangeJoin` &expr, const `ibis::bitvector` &mask, const `ibis::qRange` *const range1, const `ibis::qRange` *const range2, `ibis::bitvector64` &lower, `ibis::bitvector64` &upper) const
Evaluating a join condition with one (likely composite) index.
- virtual void `estimate` (const `ibis::rangeJoin` &expr, const `ibis::bitvector` &mask, `ibis::bitvector64` &lower, `ibis::bitvector64` &upper) const
- virtual void `estimate` (const `ibis::rangeJoin` &expr, `ibis::bitvector64` &lower, `ibis::bitvector64` &upper) const
Estimate the hits for symmetric joins.
- virtual uint32_t `estimate` (const `ibis::qContinuousRange` &expr) const
Returns an upper bound on the number of hits.
- virtual void `estimate` (const `ibis::qContinuousRange` &expr, `ibis::bitvector` &lower, `ibis::bitvector` &upper) const
Computes an approximation of hits as a pair of lower and upper bounds.
- virtual double `estimateCost` (const `ibis::qDiscreteRange` &expr) const
- virtual double `estimateCost` (const `ibis::qContinuousRange` &expr) const
Estimate the cost of evaluate a range condition.
- virtual long `evaluate` (const `ibis::qContinuousRange` &expr, `ibis::bitvector` &hits) const
To evaluate the exact hits.
- virtual int `expandRange` (`ibis::qContinuousRange` &rng) const
The functions `expandRange` and `contractRange` expands or contracts the boundaries of a range condition so that the new range will have exact answers using the function `estimate`.
- virtual long `getCumulativeDistribution` (std::vector< double > &bds, std::vector< uint32_t > &cts) const
Cumulative distribution of the data.
- virtual long `getDistribution` (std::vector< double > &bbs, std::vector< uint32_t > &cts) const

Binned distribution of the data.

- virtual double `getMax ()` const
The maximum value recorded in the index.
- virtual double `getMin ()` const
The minimum value recorded in the index.
- virtual double `getSum ()` const
Compute the approximate sum of all the values indexed.
- `array_t< uint32_t > * indices` (const `ibis::bitvector` &mask) const
- virtual const char * `name ()` const
Returns the name of the index, similar to the function `type`, but returns a string instead.
- virtual `uint32_t numBins ()` const
- virtual void `print` (std::ostream &out) const
Prints human readable information.
- void `read` (int fd, uint32_t offset, const char *fname)
Read an `ibis::bin` embedded inside a file.
- virtual void `read` (`ibis::fileManager::storage` *st)
Reconstructs an index from an array of bytes.
- virtual void `read` (const char *idxfile)
Reconstructs an index from the named file.
- virtual void `speedTest` (std::ostream &out) const
Time some logical operations and print out their speed.
- virtual `INDEX_TYPE type ()` const
Returns an index type identifier.
- virtual float `undecidable` (const `ibis::qContinuousRange` &expr, `ibis::bitvector` &iffy) const
Mark the position of the rows that can not be decided with this index.
- virtual void `write` (const char *dt) const
Save index to a file.

Protected Member Functions

- void `addBounds` (double lbd, double rbd, uint32_t nbins, uint32_t eqw)
- virtual void `adjustLength` (uint32_t nrow)
- `bin` (const `ibis::column` *c, const uint32_t nbins, `ibis::fileManager::storage` *st, uint32_t offset=8)
A constructor to handle the common portion of multicomponent encodings.
- template<typename E> void `binning` (const `array_t< E >` &v, const `array_t< double >` &bd)
- template<typename E> void `binning` (const `array_t< E >` &v)
- void `binning` (const char *f)
Read the data file and partition the values into bins according to the specified bin boundary.

- void **binning** (const char *f, const `array_t< double >` &bd)
- void **binning** (const char *f, const `std::vector< double >` &bd)

Generate bins according to the specified boundaries.
- template<typename E> void **binningT** (const char *fname)

Read the data file, partition the values, and write out the bin ordered data with .bin suffix.
- long **binOrder** (const char *fname) const
- template<typename E> long **binOrderT** (const char *fname) const

Write bin-ordered values.
- template<typename E> long **checkBin0** (const `ibis::qRange` &cmp, `uint32_t` jbin, `ibis::bitvector` &res) const
- template<typename E> long **checkBin1** (const `ibis::qRange` &cmp, `uint32_t` jbin, const `ibis::bitvector` &mask, `ibis::bitvector` &res) const
- virtual void **clear** ()

Clear the existing content.
- virtual double **computeSum** () const
- template<typename E> void **construct** (const `array_t< E >` &varr)
- void **convertGranules** (granuleMap &gmap)
- void **divideBitmaps** (const `std::vector< ibis::bitvector * >` &bms, `std::vector< unsigned >` &parts) const
- virtual void **locate** (const `ibis::qContinuousRange` &expr, `uint32_t` &cand0, `uint32_t` &cand1, `uint32_t` &hit0, `uint32_t` &hit1) const
- virtual void **locate** (const `ibis::qContinuousRange` &expr, `uint32_t` &cand0, `uint32_t` &cand1) const
- virtual `uint32_t` **locate** (const double &val) const
- template<typename E> void **mapGranules** (const `array_t< E >` &, granuleMap &gmap) const
- `uint32_t` **parseNbins** () const
- unsigned **parsePrec** () const
- unsigned **parseScale** () const
- void **printGranules** (`std::ostream` &out, const granuleMap &gmap) const
- void **readBinBoundaries** (const char *name, `uint32_t` nb)
- void **scanAndPartition** (const char *, unsigned, `uint32_t` nbins=0)

The optional argument nbins can either be set outside or set to be the return value of function parseNbins.
- template<typename E> void **scanAndPartition** (const `array_t< E >` &, unsigned)
- template<typename E> void **setBoundaries** (const `array_t< E >` &varr)
- void **setBoundaries** (`array_t< double >` &bnds, const `ibis::bin` &idx1, const `array_t< uint32_t >` cnt1, const `array_t< uint32_t >` cnt0) const
- void **setBoundaries** (`array_t< double >` &bnds, const `ibis::bin` &bin0) const
- void **setBoundaries** (const char *f)

Set bin boundaries.
- void **swap** (`bin` &rhs)

Protected Attributes

- `array_t< double >` **bounds**

The nominal boundaries.
- `array_t< double >` **maxval**

The maximal values in each bin.
- `array_t< double >` **minval**

The minimal values in each bin.

- `uint32_t nobs`

Number of bitvectors.

Friends

- class `ibis::ambit`
- class `ibis::band`
- class `ibis::mesa`
- class `ibis::mesh`
- class `ibis::pack`
- class `ibis::pale`
- class `ibis::range`
- class `ibis::zone`

Classes

- struct `granule`

A data structure to assist the mapping of values to lower precisions.

3.7.1 Detailed Description

The equality encoded bitmap index with binning.

The exact bin boundary assignment is controlled by indexing options '`<binning ... />`'.

The 0th bit vector represents $x < \text{bounds}[0]$; The (nobs-1)st vector represents $x \geq \text{bounds}[\text{nobs}-2]$; The *i*th bit vector represents $\text{bounds}[i-1] \leq x < \text{bounds}[i]$, ($0 < i < \text{nobs}-1$).

3.7.2 Member Function Documentation

3.7.2.1 `void ibis::bin::binning (const char *f, const std::vector< double > &bd) [protected]`

Generate bins according to the specified boundaries.

Note:

This function does not attempt to clear the content of the current data structure, the caller is responsible for this task!

3.7.2.2 `long ibis::bin::checkBin (const ibis::qRange &cmp, uint32_t jbin, const ibis::bitvector &mask, ibis::bitvector &res) const`

Candidate check using the binned values.

The bitvector `mask` marks the actual values in the bin (because the bitmaps stored in `bits` do not directly corresponds to the bin).

3.7.2.3 `long ibis::bin::checkBin (const ibis::qRange &cmp, uint32_t jbin, ibis::bitvector &res) const`

Candidate check using the binned values.

Returns the number of hits if successful, otherwise it returns a negative value.

3.7.2.4 `void ibis::bin::estimate (const ibis::rangeJoin & expr, ibis::bitvector64 & lower, ibis::bitvector64 & upper) const` `[virtual]`

Estimate the hits for symmetric joins.

Record the definite hits in `lower`, and all possible hits in `upper`. NOTE: `upper` includes all entries in `lower`.

3.7.2.5 `void ibis::bin::estimate (const ibis::qContinuousRange & expr, ibis::bitvector & lower, ibis::bitvector & upper) const` `[virtual]`

Computes an approximation of hits as a pair of lower and upper bounds.

Parameters:

expr the query expression to be evaluated.

lower a bitvector marking a subset of the hits. All rows marked with one (1) are definitely hits.

upper a bitvector marking a superset of the hits. All hits are marked with one, but some of the rows marked one may not be hits. If the variable `upper` is empty, the variable `lower` is assumed to contain the exact answer.

Implements [ibis::index](#).

Reimplemented in [ibis::range](#), [ibis::mesa](#), [ibis::ambit](#), [ibis::pale](#), [ibis::pack](#), [ibis::zone](#), [ibis::egale](#), [ibis::moins](#), and [ibis::entre](#).

3.7.2.6 `long ibis::bin::evaluate (const ibis::qContinuousRange & expr, ibis::bitvector & hits) const` `[virtual]`

To evaluate the exact hits.

On success, return the number of hits, otherwise a negative value is returned.

Implements [ibis::index](#).

Reimplemented in [ibis::range](#), [ibis::mesa](#), [ibis::ambit](#), [ibis::pale](#), [ibis::pack](#), [ibis::zone](#), [ibis::egale](#), [ibis::moins](#), and [ibis::entre](#).

3.7.2.7 `int ibis::bin::expandRange (ibis::qContinuousRange & rng) const` `[virtual]`

The functions `expandRange` and `contractRange` expands or contracts the boundaries of a range condition so that the new range will have exact answers using the function estimate.

The default implementation provided does nothing since this is only meaningful for indices based on bins.

Reimplemented from [ibis::index](#).

Reimplemented in [ibis::range](#), [ibis::bak](#), and [ibis::bak2](#).

3.7.2.8 `double ibis::bin::getSum () const` `[virtual]`

Compute the approximate sum of all the values indexed.

If it decides that computing the sum directly from the vertical partition is more efficient, it will return NaN immediately.

Implements [ibis::index](#).

Reimplemented in [ibis::range](#), [ibis::mesa](#), [ibis::ambit](#), [ibis::pack](#), [ibis::egale](#), [ibis::moins](#), and [ibis::entre](#).

3.7.2.9 `void ibis::bin::print (std::ostream & out) const` `[virtual]`

Prints human readable information.

Outputs information about the index as text to the specified output stream.

Implements [ibis::index](#).

Reimplemented in [ibis::range](#), [ibis::mesa](#), [ibis::ambit](#), [ibis::pale](#), [ibis::pack](#), [ibis::zone](#), [ibis::egale](#), [ibis::moins](#), [ibis::entre](#), [ibis::bak](#), and [ibis::bak2](#).

3.7.2.10 `void ibis::bin::read (int fdes, uint32_t start, const char * fn)`

Read an [ibis::bin](#) embedded inside a file.

This is intended to be used by multi-level indices.

Reimplemented in [ibis::range](#).

3.7.2.11 `void ibis::bin::read (ibis::fileManager::storage * st)` [virtual]

Reconstructs an index from an array of bytes.

Intended for internal use only!

Implements [ibis::index](#).

Reimplemented in [ibis::range](#), [ibis::ambit](#), [ibis::pale](#), [ibis::pack](#), [ibis::zone](#), and [ibis::egale](#).

3.7.2.12 `void ibis::bin::read (const char * idxfile)` [virtual]

Reconstructs an index from the named file.

The name can be the directory containing an index file. In this case, the name of the index file must be the name of the column followed by ".idx" suffix.

Implements [ibis::index](#).

Reimplemented in [ibis::range](#), [ibis::ambit](#), [ibis::pale](#), [ibis::pack](#), [ibis::zone](#), [ibis::egale](#), [ibis::bak](#), and [ibis::bak2](#).

3.7.2.13 `void ibis::bin::setBoundaries (const char * f)` [protected]

Set bin boundaries.

The bin specification can be of the following, where all fields are optional.

- `equal ([_ -]?) [weight|length|ratio]`
- `no=xxx|nbins=xxx|bins:(\[begin, end, no=xxx\])+`
- `<binning (start=begin end=end nbins=xxx scale=[linear|log])* />`
- `<binning binFile=file-name[, nbins=xxx] />`

The bin specification can be read from the column object, the table object containing the column, or the global [ibis::g-Parameters](#) object under the name of `table-name.column-name.index`. If no index specification is found, it builds approximate equal weight bins.

Note:

If equal weight is specified, it take precedence over all other specification.

3.7.2.14 `float ibis::bin::undecidable (const ibis::qContinuousRange & expr, ibis::bitvector & iffy)` const [virtual]

Mark the position of the rows that can not be decided with this index.

Parameters:

expr the range conditions to be evaluated.

iffy the bitvector marking the positions of rows that can not be decided using the index. Return value is the expected fraction of undecided rows that might satisfy the range conditions.

Implements [ibis::index](#).

Reimplemented in [ibis::range](#), [ibis::mesa](#), [ibis::ambit](#), [ibis::pale](#), [ibis::pack](#), [ibis::zone](#), and [ibis::egale](#).

3.7.2.15 `void ibis::bin::write (const char * dt) const` [virtual]

Save index to a file.

Outputs the index in a compact binary format to the named file or directory. The index file contains a header that can be identified by the function `isIndex`.

Implements [ibis::index](#).

Reimplemented in [ibis::range](#), [ibis::mesa](#), [ibis::ambit](#), [ibis::pale](#), [ibis::pack](#), [ibis::zone](#), [ibis::egale](#), [ibis::moins](#), [ibis::entre](#), [ibis::bak](#), and [ibis::bak2](#).

The documentation for this class was generated from the following files:

- [ibin.h](#)
- [ibin.cpp](#)

3.8 `ibis::bin::granule` Struct Reference

A data structure to assist the mapping of values to lower precisions.

```
#include <ibin.h>
```

Public Attributes

- `ibis::bitvector * loc0`
- `ibis::bitvector * loc1`
- double `max0`
- double `max1`
- double `min0`
- double `min1`

3.8.1 Detailed Description

A data structure to assist the mapping of values to lower precisions.

Any integral or floating-point value may be mapped to lower precision floating-point value. This would produce a more granular representation of the values. The low precision floating-point value is called a target in this description. To facilitate this type of dynamic binning, we device this simple data structure to record the position of all records mapped to a particular target value. It separates out the values that are larger than or equal to the target from those that are smaller than the target. The variables `min0` and `max0` store the actual minimum and maximum value among those that are smaller than the target. The variables `min1` and `max1` store the actual minimum and maximum value among those that are larger than or equal to the target value.

The documentation for this struct was generated from the following file:

- [ibin.h](#)

3.9 `ibis::bitvector` Class Reference

A data structure to represent a sequence of bits.

```
#include <bitvector.h>
```

Public Types

- typedef `uint32_t` `word_t`
The basic unit of data storage.

Public Member Functions

- void `adjustSize` (`word_t` `nv`, `word_t` `nt`)
Adjust the size of the bit sequence.
- void `appendFill` (`int` `val`, `word_t` `n`)
Append `n` bits of `val`.
- void `appendWord` (`word_t` `w`)
Append a word of bits.
- `const_iterator` `begin` () const
- `iterator` `begin` ()
- `bitvector` (`const char` *`file`)
Read the content of the named file.
- `bitvector` (`const array_t`< `word_t` > &`arr`)
Construct a bitvector from an array.
- `bitvector` (`const bitvector` &`bv`)
Shallow copy. Underlying storage is reference counted.
- `uint32_t` `bytes` () const throw ()
Return the number of bytes used by the bitvector object in memory.
- void `clear` ()
Remove the existing content of a bitvector.
- `word_t` `cnt` () const
Return the number of bits that are one.
- void `compress` ()
Merge fills into fill words.
- `word_t` `compressible` () const
Return the number of word saved if the function `compress` is called.
- `bitvector` & `copy` (`const bitvector` &`bv`)
Note:
Deep copy.

- void `decompress` ()
Turn all fill words into literal words.
- `const_iterator end` () const
- `iterator end` ()
- void `erase` (`word_t` i, `word_t` j)
Remove the bits in the range of [i, j).
- `indexSet firstIndexSet` () const
- void `flip` ()
Bitwise operations.
- `uint32_t getSerialSize` () const throw ()
Compute the number of word in serialized version of the bitvector object.
- `word_t numFillWords` () const
Return the number of fill words.
- `bitvector * operator &` (const `bitvector &`) const
- void `operator &=` (const `bitvector &`rhs)
Perform bitwise AND between this bitvector and rhs.
- `bitvector & operator+=` (int b)
Append a single bit.
- `bitvector & operator+=` (const `bitvector &`bv)
Append a bitvector.
- `bitvector * operator-` (const `bitvector &`) const
- void `operator-=` (const `bitvector &`rhs)
Perform bitwise subtraction (a & !b).
- const `bitvector & operator=` (const `bitvector &`bv)
Note:
Deep copy.
- int `operator==` (const `bitvector &`rhs) const
- `bitvector * operator^` (const `bitvector &`) const
- void `operator^=` (const `bitvector &`rhs)
Perform bitwise exclusive or (XOR).
- `bitvector * operator|` (const `bitvector &`) const
- void `operator|=` (const `bitvector &`rhs)
Perform bitwise OR.
- `std::ostream & print` (`std::ostream &`) const
The print function.
- void `read` (const char *fn)
I/O functions.
- void `reserve` (unsigned nb, unsigned nc, double cf=0.0)
- void `set` (int val, `word_t` n)

Create a vector with `n` bits of value `val` (cf.

- void `setBit` (const `word_t` `i`, int `val`)
Replace a single bit at position `i`.
- void `setSize` (`word_t` `n`) const
Explicitly set the size of the bitvector.
- `word_t` `size` () const throw ()
Return the total number of bits in the bit sequence.
- `bitvector` & `swap` (`bitvector` &`bv`)
- void `turnOnRawBit` (const `word_t` `i`)
Turn on a single bit in a uncompressed bitvector.
- void `write` (`array_t`< `word_t` > &`arr`) const
Write the bit vector to an `array_t`<`word_t`>.
- void `write` (int `fdes`) const
Write to a file that is opened by the caller.
- void `write` (const char *`fn`) const
Write the bit vector to a file.

Static Public Member Functions

- static `word_t` `bitsPerLiteral` ()
Return the number of bits in a literal word.
- static double `clusteringFactor` (`word_t` `nb`, `word_t` `nc`, `word_t` `sz`)
Estimate clustering factor based on the size.
- static double `markovSize` (`word_t` `nb`, `word_t` `nc`, double `f`)
Compute the expected size (number of bytes) of a random sequence generated from a Markov process.
- static double `randomSize` (`word_t` `nb`, `word_t` `nc`)
Compute the expected number of bytes required to store a random sequence.

Protected Member Functions

- bool `all0s` () const
Are all bits in regular words 0?
- bool `all1s` () const
Are all bits in regular words 1?
- bool `isCompressed` () const

Friends

- struct `active_word`
- class `const_iterator`
- class `indexSet`
- class `iterator`
- struct `run`

Classes

- struct `active_word`
The struct `active_word` stores the last few bits that do not fill a whole word.
- class `const_iterator`
The `const_iterator` class. It iterates on the individual bits.
- class `indexSet`
The `indexSet` stores positions of bits that are one.
- class `iterator`
The iterator that allows modification of bits.
- struct `run`
An internal struct used during logical operations to track the usage of fill words.

3.9.1 Detailed Description

A data structure to represent a sequence of bits.

Key features

A bitvector object stores a sequence of bits and provides fast bitwise logical operations. In addition, it supports operations to append new bits from the end, read bits at arbitrary location and set bits at arbitrary location. It also supports an iterator, a `const_iterator` and an `indexSet`.

Encoding format

Incoming bits are organized into words (`bitvector::word_t`). A word is a literal word if its Most Significant Bit (MSB) is 0, it is a fill word if its MSB is 1. A literal word stores literal bit values in the bit position following the MSB and a fill word stores a sequence of consecutive bits that are of the same value, i.e., a fill. The second most significant bit of the fill word is the bit value, the remaining bits of the word is a unsigned integer that stores the length of the fill as number of equivalent literal words, i.e., how many literal words it will take if the fill is stored in literal words.

Restrictions

- The number of bits must be expressible by one single `bitvector::word_t`. This ensure that a fill word can store a fill of any valid length without performing a bound check. If `bitvector::word_t` is 32-bit long, the maximum number of bits that can be represented by a bitvector object is 4 billion.
- When adding a bit with `bitvector::operator+=`, the integer value passed in must be one of 0 or 1. Since checking whether the import value is 0 or not 0 causes pipeline bubble in CPU, we have opted for not performing the check. An input value other than 0 or 1 will cause existing bits to be modified in unpredictable ways.

3.9.2 Constructor & Destructor Documentation

3.9.2.1 `ibis::bitvector::bitvector (const array_t< word_t > & arr)` [inline]

Construct a bitvector from an array.

Because the array copy constructor performs shallow copy, this bitvector is not using any new space.

3.9.3 Member Function Documentation

3.9.3.1 `void ibis::bitvector::adjustSize (word_t nv, word_t nt)`

Adjust the size of the bit sequence.

If current size is less than `nv`, append enough 1 bits so that it has `nv` bits. If the resulting total number of bits is less than `nt`, append 0 bits so that there are `nt` total bits. The final result always contains `nt` bits.

3.9.3.2 `double ibis::bitvector::clusteringFactor (word_t nb, word_t nc, word_t sz)` [static]

Estimate clustering factor based on the size.

The size is measured as the number of bytes. [markovSize](#).

3.9.3.3 `void ibis::bitvector::flip ()`

Bitwise operations.

Complement all bits of a bit sequence.

3.9.3.4 `uint32_t ibis::bitvector::getSerialSize () const throw ()` [inline]

Compute the number of word in serialized version of the bitvector object.

This would be the size of this bitvector object on disk or in a single `array_t<word_t>`.

3.9.3.5 `double ibis::bitvector::markovSize (word_t nb, word_t nc, double f)` [inline, static]

Compute the expected size (number of bytes) of a random sequence generated from a Markov process.

The bit sequence is to have `nb` total bits, `nc` bits of one, and `f` consecutive ones on the average. The argument `f` is known as the clustering factor.

3.9.3.6 `const ibis::bitvector & ibis::bitvector::operator= (const bitvector & bv)` [inline]

Note:

Deep copy.

Use deep copy. Wanted to use shallow copy for efficiency considerations, but SHALLOW copy causes unexpected problem in test program `bitty.cpp`.

3.9.3.7 `double ibis::bitvector::randomSize (word_t nb, word_t nc)` [inline, static]

Compute the expected number of bytes required to store a random sequence.

The random bit sequence is to have `nb` total bits and `nc` bits of one.

3.9.3.8 `void ibis::bitvector::read (const char *fn)`

I/O functions.

Read a bit vector from the file. Purge current contents before read.

3.9.3.9 `void ibis::bitvector::set (int val, word_t n)`

Create a vector with `n` bits of value `val` (cf. `memset()`).

Note:

`val` must be either 0 or 1.

3.9.3.10 `void ibis::bitvector::setBit (const word_t i, int val)`

Replace a single bit at position `i`.

Note:

`val` must be either 0 or 1.

3.9.3.11 `void ibis::bitvector::setSize (word_t n) const [inline]`

Explicitly set the size of the bitvector.

Caller is responsible for ensuring the size assigned is actually correct.

3.9.3.12 `void ibis::bitvector::write (array_t< word_t > &arr) const`

Write the bit vector to an `array_t<word_t>`.

The serialize version of the bit vector may be passed to another I/O function or sent through networks.

The documentation for this class was generated from the following files:

- [bitvector.h](#)
- [bitvector.cpp](#)

3.10 `ibis::bitvector64` Class Reference

A data structure to represent a sequence of bits.

```
#include <bitvector64.h>
```

Public Types

- typedef uint64_t `word_t`

The basic unit of data storage is 64-bit.

Public Member Functions

- void `adjustSize (word_t nv, word_t nt)`

Adjust the size of the bit sequence.

- void `appendFill` (int val, `word_t` n)
Append n bits of val.
- void `appendWord` (`word_t` w)
Append a word of bits.
- `const_iterator` `begin` () const
- `iterator` `begin` ()
- `bitvector64` (const char *file)
Read the content of the named file.
- `bitvector64` (const `array_t`< `word_t` > &arr)
- `bitvector64` (const `bitvector64` &bv)
- `word_t` `bytes` () const throw ()
Return the number of bytes used by the bitvector object in memory.
- void `clear` ()
Remove the existing content of a `bitvector64`.
- `word_t` `cnt` () const
Return the number of bits that are one.
- void `compress` ()
Merge fills into fill words.
- `word_t` `compressible` () const
Return the number of word saved if the function compress is called.
- `bitvector64` & `copy` (const `bitvector64` &bv)
Note:
Deep copy.
- void `decompress` ()
Turn all fill words into literal words.
- `const_iterator` `end` () const
- `iterator` `end` ()
- void `erase` (`word_t` i, `word_t` j)
Remove the bits in the range of [i, j).
- `indexSet` `firstIndexSet` () const
- void `flip` ()
Bitwise operations.
- `word_t` `getSerialSize` () const throw ()
Compute the number of words in serialized version of the bitvector object.
- `word_t` `numFillWords` () const
Return the number of fill words.
- `bitvector64` * `operator &` (const `bitvector64` &) const
- void `operator &=` (const `bitvector64` &rhs)

Perform bitwise AND between this `bitvector64` and `rhs`.

- `bitvector64 & operator+= (int b)`
Append a single bit.
- `bitvector64 & operator+= (const bitvector64 &bv)`
Append a `bitvector64`.
- `bitvector64 * operator- (const bitvector64 &) const`
- `void operator-= (const bitvector64 &rhs)`
Perform bitwise subtraction ($a \& !b$).
- `bitvector64 & operator= (const bitvector64 &bv)`
Note:
Deep copy.
- `int operator== (const bitvector64 &rhs) const`
- `bitvector64 * operator^ (const bitvector64 &) const`
- `void operator^= (const bitvector64 &rhs)`
Perform bitwise exclusive or (XOR).
- `bitvector64 * operator| (const bitvector64 &) const`
- `void operator|= (const bitvector64 &rhs)`
Perform bitwise OR.
- `std::ostream & print (std::ostream &) const`
The print function.
- `void read (const char *fn)`
I/O functions.
- `void set (int val, word_t n)`
Create a vector with `n` bits of value `val` (cf.
- `void setBit (const word_t i, int val)`
Replace a single bit at position `i`.
- `word_t size () const throw ()`
Return the total number of bits in the bit sequence.
- `bitvector64 & swap (bitvector64 &bv)`
- `void write (array_t< word_t > &arr) const`
Write the bit vector to an `array_t<word_t>`.
- `void write (FILE *fptr) const`
- `void write (const char *fn) const`
Write the bit vector to a file.

Static Public Member Functions

- static unsigned `bitsPerLiteral` ()
Return the number of bits in a literal word.
- static double `clusteringFactor` (`word_t` nb, `word_t` nc, `word_t` nw)
Estimate clustering factor based on the size.
- static double `markovSize` (`word_t` nb, `word_t` nc, double f)
Compute the expected size (bytes) of a random sequence generated from a Markov process.
- static double `randomSize` (`word_t` nb, `word_t` nc)
Compute the expected number of bytes required to store a random sequence.

Protected Member Functions

- bool `all0s` () const
Are all bits in regular words 0?
- bool `all1s` () const
Are all bits in regular words 1?
- bool `isCompressed` () const

Friends

- struct `active_word`
- class `const_iterator`
- class `indexSet`
- class `iterator`
- struct `run`

Classes

- struct `active_word`
The struct `active_word` stores the last few bits that do not fill a whole word.
- class `const_iterator`
The `const_iterator` class. It iterates on the individual bits.
- class `indexSet`
The `indexSet` stores positions of bits that are one.
- class `iterator`
The iterator that allows modification of bits.
- struct `run`
An internal struct used during logical operations to track the usage of fill words.

3.10.1 Detailed Description

A data structure to represent a sequence of bits.

The 64-bit version.

Key features

A bitvector object stores a sequence of bits and provides fast bitwise logical operations. In addition, it supports operations to append new bits from the end, read bits at arbitrary location and set bits at arbitrary location. It also supports an iterator, a [const_iterator](#) and an [indexSet](#).

Encoding format

Incoming bits are organized into words (`bitvector::word_t`). A word is a literal word if its Most Significant Bit (MSB) is 0, it is a fill word if its MSB is 1. A literal word stores literal bit values in the bit position following the MSB and a fill word stores a sequence of consecutive bits that are of the same value, i.e., a fill. The second most significant bit of the fill word is the bit value, the remaining bits of the word is a unsigned integer that stores the length of the fill as number of equivalent literal words, i.e., how many literal words it will take if the fill is stored in literal words.

Restrictions

- The number of bits must be expressible by one single `bitvector::word_t`. This ensure that a fill word can store a fill of any valid length without performing a bound check. In this 64-bit version, the maximum number of bits that can be represented by a bitvector object is 16 quintillion (16×10^{18}).
- When adding a bit with `bitvector::operator+=`, the integer value passed in must be one of 0 or 1. Since checking whether the import value is 0 or not 0 causes pipeline bubble in CPU, we have opted for not performing the check. An input value other than 0 or 1 will cause existing bits to be modified in unpredictable ways.

3.10.2 Member Function Documentation

3.10.2.1 `void ibis::bitvector64::adjustSize (word_t nv, word_t nt)`

Adjust the size of the bit sequence.

If current size is less than `nv`, append enough 1 bits so that it has `nv` bits. If the resulting total number of bits is less than `nt`, append 0 bits so that there are `nt` total bits. The final result always contains `nt` bits.

3.10.2.2 `double ibis::bitvector64::clusteringFactor (word_t nb, word_t nc, word_t nw) [static]`

Estimate clustering factor based on the size.

[markovSize](#).

3.10.2.3 `void ibis::bitvector64::flip ()`

Bitwise operations.

Complement all bits of a bit sequence.

3.10.2.4 `double ibis::bitvector64::markovSize (word_t nb, word_t nc, double f) [inline, static]`

Compute the expected size (bytes) of a random sequence generated from a Markov process.

The bit sequence is to have `nb` total bits, `nc` bits of one, and `f` consecutive ones on the average. The argument `f` is known as the clustering factor.

3.10.2.5 `double ibis::bitvector64::randomSize (word_t nb, word_t nc) [inline, static]`

Compute the expected number of bytes required to store a random sequence.

The random bit sequence is to have `nb` total bits and `nc` bits of one.

3.10.2.6 `void ibis::bitvector64::read (const char *fn)`

I/O functions.

Read a bit vector from the file. Purge current contents before read.

3.10.2.7 `void ibis::bitvector64::set (int val, word_t n)`

Create a vector with `n` bits of value `val` (cf. `memset()`).

Note:

`val` must be either 0 or 1.

3.10.2.8 `void ibis::bitvector64::setBit (const word_t i, int val)`

Replace a single bit at position `i`.

Note:

`val` must be either 0 or 1.

The documentation for this class was generated from the following files:

- [bitvector64.h](#)
- [bitvector64.cpp](#)

3.11 `ibis::bitvector64::const_iterator` Class Reference

The `const_iterator` class. It iterates on the individual bits.

```
#include <bitvector64.h>
```

Public Member Functions

- `bool operator * () const`
- `int operator!= (const const_iterator &rhs) const throw ()`
- `const_iterator & operator++ ()`
- `const_iterator & operator+= (int64_t incr)`
- `const_iterator & operator- ()`
- `int operator== (const const_iterator &rhs) const throw ()`

Friends

- `const_iterator ibis::bitvector64::begin () const`
- `const_iterator ibis::bitvector64::end () const`
- class `ibis::bitvector64::iterator`

3.11.1 Detailed Description

The `const_iterator` class. It iterates on the individual bits.

The documentation for this class was generated from the following files:

- [bitvector64.h](#)
- [bitvector64.cpp](#)

3.12 `ibis::bitvector64::indexSet` Class Reference

The `indexSet` stores positions of bits that are one.

```
#include <bitvector64.h>
```

Public Member Functions

- const `word_t * indices` () const
- bool `isRange` () const
- `word_t nIndices` () const
- `indexSet & operator++` ()

Friends

- `indexSet ibis::bitvector64::firstIndexSet` () const

3.12.1 Detailed Description

The `indexSet` stores positions of bits that are one.

It decodes one word of the `bitvector64` as a time. For a fill of ones, the function `isRange` returns true, otherwise it returns false. If `isRange` returns true, the position of the first bit is pointed by the pointer returned by function `indices`, and there are `nIndices` consecutive ones. If `isRange` returns false, there are `nIndices` bits that are one and the positions of these bits are stored in the array returned by function `indices`.

The documentation for this class was generated from the following files:

- `bitvector64.h`
- `bitvector64.cpp`

3.13 `ibis::bitvector64::iterator` Class Reference

The iterator that allows modification of bits.

```
#include <bitvector64.h>
```

Public Member Functions

- bool `operator *` () const
- int `operator!==(const iterator &rhs)` const throw ()
- int `operator!==(const const_iterator &rhs)` const throw ()
- `iterator & operator++` ()
- `iterator & operator+=(int64_t incr)`
- `iterator & operator--` ()
- `iterator & operator=(int val)`
- int `operator==(const iterator &rhs)` const throw ()
- int `operator==(const const_iterator &rhs)` const throw ()

Friends

- `iterator ibis::bitvector64::begin` ()
- `iterator ibis::bitvector64::end` ()
- void `ibis::bitvector64::erase` (`word_t i`, `word_t j`)

3.13.1 Detailed Description

The iterator that allows modification of bits.

It provides only one additional function (`operator=`) than `const_iterator` to allow modification of the bit pointed.

IMPORTANT: `operator=` modifies the content of the `bitvector64` it points to and it can invalidate other iterator or `const_iterator` referring to the same `bitvector64`.

The documentation for this class was generated from the following files:

- [bitvector64.h](#)
- [bitvector64.cpp](#)

3.14 `ibis::bitvector::const_iterator` Class Reference

The `const_iterator` class. It iterates on the individual bits.

```
#include <bitvector.h>
```

Public Member Functions

- `const_iterator` (const `const_iterator` &r)
- `bool operator * ()` const
- `int operator!= (const const_iterator &rhs)` const throw ()
- `const_iterator & operator++ ()`
- `const_iterator & operator+= (int incr)`
- `const_iterator & operator- ()`
- `const_iterator & operator= (const const_iterator &r)`
- `int operator== (const const_iterator &rhs)` const throw ()

Friends

- `const_iterator` `ibis::bitvector::begin ()` const
- `const_iterator` `ibis::bitvector::end ()` const
- class `ibis::bitvector::iterator`

3.14.1 Detailed Description

The `const_iterator` class. It iterates on the individual bits.

The documentation for this class was generated from the following files:

- [bitvector.h](#)
- [bitvector.cpp](#)

3.15 `ibis::bitvector::indexSet` Class Reference

The `indexSet` stores positions of bits that are one.

```
#include <bitvector.h>
```

Public Member Functions

- `const word_t * indices () const`
- `bool isRange () const`
- `word_t nIndices () const`
- `indexSet & operator++ ()`

Friends

- `indexSet ibis::bitvector::firstIndexSet () const`

3.15.1 Detailed Description

The `indexSet` stores positions of bits that are one.

It decodes one word of the bitvector at a time. For a fill of ones, the function `isRange` returns true, otherwise it returns false. If `isRange` returns true, the position of the first bit is pointed by the pointer returned by function `indices`, and there are `nIndices` consecutive ones. If `isRange` returns false, there are `nIndices` bits that are one and the positions of these bits are stored in the array returned by function `indices`.

The documentation for this class was generated from the following files:

- `bitvector.h`
- `bitvector.cpp`

3.16 `ibis::bitvector::iterator` Class Reference

The iterator that allows modification of bits.

```
#include <bitvector.h>
```

Public Member Functions

- `iterator (const iterator &r)`
- `bool operator * () const`
- `int operator!= (const iterator &rhs) const throw ()`
- `int operator!= (const const_iterator &rhs) const throw ()`
- `iterator & operator++ ()`
- `iterator & operator+= (int incr)`
- `iterator & operator- ()`
- `const iterator & operator= (int val)`
- `const iterator & operator= (const iterator &r)`
- `int operator== (const iterator &rhs) const throw ()`
- `int operator== (const const_iterator &rhs) const throw ()`

Friends

- `iterator ibis::bitvector::begin ()`
- `iterator ibis::bitvector::end ()`
- `void ibis::bitvector::erase (word_t i, word_t j)`

3.16.1 Detailed Description

The iterator that allows modification of bits.

It provides only one additional function (operator=) than `const_iterator` to allow modification of the bit pointed.

IMPORTANT: operator= modifies the content of the bitvector it points to and it can invalidate other iterators or `const_` iterators referring to the same bitvector.

The documentation for this class was generated from the following files:

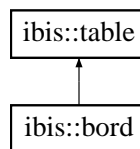
- [bitvector.h](#)
- [bitvector.cpp](#)

3.17 `ibis::bord` Class Reference

Class `ibis::bord` stores all its data in memory.

```
#include <bord.h>
```

Inheritance diagram for `ibis::bord`:



Public Types

- typedef `std::vector< void * >` **bufferList**

Public Member Functions

- **bord** (`const char *tn, const char *td, uint64_t nr, const ibis::table::stringList &cn, const ibis::table::typeList &ct, const bufferList &buf`)
- virtual int [buildIndex](#) (`const char *, const char *`)
Create the index for named column.
- virtual int [buildIndexes](#) (`const char *`)
Create indexes for every column of the table.
- virtual [ibis::table::stringList](#) [columnNames](#) () const
Return column names.
- virtual [ibis::table::typeList](#) [columnTypes](#) () const
Return data types.
- virtual `ibis::table::cursor *` [createCursor](#) () const
Create a `CURSOR` object to perform row-wise data access.
- virtual void [describe](#) (`std::ostream &`) const
Print a description of the table to the specified output stream.
- virtual int [dump](#) (`std::ostream &, const char *`) const

Dump the values in ASCII form to the specified output stream.

- virtual void [estimate](#) (const char *cond, uint64_t &nmin, uint64_t &nmax) const
Estimate the number of rows satisfying the selection conditions.
- virtual int64_t [getColumnAsBytes](#) (const char *, char *) const
Retrieve all values of the named column.
- virtual int64_t [getColumnAsDoubles](#) (const char *, double *) const
Retrieve all values of the named column.
- virtual int64_t [getColumnAsFloats](#) (const char *, float *) const
Retrieve all values of the named column.
- virtual int64_t [getColumnAsInts](#) (const char *, int32_t *) const
Retrieve all values of the named column.
- virtual int64_t [getColumnAsLongs](#) (const char *, int64_t *) const
Retrieve all values of the named column.
- virtual int64_t [getColumnAsShorts](#) (const char *, int16_t *) const
Retrieve all values of the named column.
- virtual int64_t [getColumnAsStrings](#) (const char *, std::vector< std::string > &) const
Retrieve the null-terminated strings as a vector of std::string objects.
- virtual int64_t [getColumnAsUBytes](#) (const char *, unsigned char *) const
Retrieve all values of the named column.
- virtual int64_t [getColumnAsUInts](#) (const char *, uint32_t *) const
Retrieve all values of the named column.
- virtual int64_t [getColumnAsULongs](#) (const char *, uint64_t *) const
Retrieve all values of the named column.
- virtual int64_t [getColumnAsUShorts](#) (const char *, uint16_t *) const
Retrieve all values of the named column.
- virtual long [getHistogram](#) (const char *, const char *, double, double, double, std::vector< size_t > &) const
Compute the histogram of the named column.
- virtual long [getHistogram2D](#) (const char *, const char *, double, double, double, const char *, double, double, double, std::vector< size_t > &) const
Compute a two-dimension histogram on columns `cname1` and `name2`.
- virtual long [getHistogram3D](#) (const char *, const char *, double, double, double, const char *, double, double, double, const char *, double, double, double, std::vector< size_t > &) const
Compute a three-dimensional histogram on the named columns.
- virtual `table *` [groupby](#) (const [ibis::table::stringList](#) &) const
Perform aggregate functions on the current table.
- virtual void [indexSpec](#) (const char *, const char *)

Replace the current indexing option.

- virtual const char * **indexSpec** (const char *) const

Retrieve the current indexing option.

- virtual size_t **nColumns** () const
- virtual uint64_t **nRows** () const
- virtual void **orderBy** (const **ibis::table::stringList** &)

Reorder the rows.

- virtual void **reverseRows** ()

Reverse the order of the rows.

- virtual **table** * **select** (const char *sel, const char *cond) const

Given a set of column names and a set of selection conditions, compute another table that represents the selected values.

Protected Member Functions

- void **clear** ()
Clear the existing content.
- int64_t **computeHits** (const char *cond) const
Compute the number of hits.

Protected Attributes

- part **mypart**

Friends

- class **cursor**

Classes

- class **column**
*An in-memory version of **ibis::column**.*
- class **cursor**
- class **part**

3.17.1 Detailed Description

Class **ibis::bord** stores all its data in memory.

The function **ibis::table::select** produces an **ibis::bord** object if the query produce nontrivial results.

Note:

Bord is the Danish word for "table."

3.17.2 Member Function Documentation

3.17.2.1 `virtual int ibis::bord::buildIndex (const char *, const char *)` `[inline, virtual]`

Create the index for named column.

The existing index will be replaced. If an indexing option is not specified, it will use the internally recorded option for the named column or the table containing the column.

Note:

Unless any there is a specific instruction to not index a column, the querying functions will automatically build indices as necessary. However, as building an index is relatively expensive process, building an index on a column is on average about four or five times as expensive as reading the column from disk, this function is provided so that it is possible to build indexes beforehand.

Implements [ibis::table](#).

3.17.2.2 `virtual int ibis::bord::buildIndexes (const char *)` `[inline, virtual]`

Create indexes for every column of the table.

Existing indexes will be replaced. If an indexing option is not specified, the internally recorded options will be used. [buildIndex](#)

Implements [ibis::table](#).

3.17.2.3 `int ibis::bord::dump (std::ostream &, const char *) const` `[inline, virtual]`

Dump the values in ASCII form to the specified output stream.

The default delimiter is coma (","), which produces Comma-Separated-Values (CSV).

Implements [ibis::table](#).

3.17.2.4 `void ibis::bord::estimate (const char * cond, uint64_t & nmin, uint64_t & nmax) const` `[virtual]`

Estimate the number of rows satisfying the selection conditions.

The number of rows is between `[nmin, nmax]`.

Implements [ibis::table](#).

3.17.2.5 `int64_t ibis::bord::getColumnAsBytes (const char *, char *) const` `[virtual]`

Retrieve all values of the named column.

The member functions of this class only support access to whole column at a time. Use `table::cursor` class for row-wise accesses. For fixed-width data types, the raw pointers are used to point to the values to be returned. In these cases, the caller is responsible for allocating enough storage for the values to be returned.

Implements [ibis::table](#).

3.17.2.6 `int64_t ibis::bord::getColumnAsDoubles (const char *, double *) const` `[virtual]`

Retrieve all values of the named column.

The member functions of this class only support access to whole column at a time. Use `table::cursor` class for row-wise accesses. For fixed-width data types, the raw pointers are used to point to the values to be returned. In these cases, the caller is responsible for allocating enough storage for the values to be returned.

Implements [ibis::table](#).

3.17.2.7 `int64_t ibis::bord::getColumnAsFloats (const char *, float *) const` [virtual]

Retrieve all values of the named column.

The member functions of this class only support access to whole column at a time. Use `table::cursor` class for row-wise accesses. For fixed-width data types, the raw pointers are used to point to the values to be returned. In these cases, the caller is responsible for allocating enough storage for the values to be returned.

Implements `ibis::table`.

3.17.2.8 `int64_t ibis::bord::getColumnAsInts (const char *, int32_t *) const` [virtual]

Retrieve all values of the named column.

The member functions of this class only support access to whole column at a time. Use `table::cursor` class for row-wise accesses. For fixed-width data types, the raw pointers are used to point to the values to be returned. In these cases, the caller is responsible for allocating enough storage for the values to be returned.

Implements `ibis::table`.

3.17.2.9 `int64_t ibis::bord::getColumnAsLongs (const char *, int64_t *) const` [virtual]

Retrieve all values of the named column.

The member functions of this class only support access to whole column at a time. Use `table::cursor` class for row-wise accesses. For fixed-width data types, the raw pointers are used to point to the values to be returned. In these cases, the caller is responsible for allocating enough storage for the values to be returned.

Implements `ibis::table`.

3.17.2.10 `int64_t ibis::bord::getColumnAsShorts (const char *, int16_t *) const` [virtual]

Retrieve all values of the named column.

The member functions of this class only support access to whole column at a time. Use `table::cursor` class for row-wise accesses. For fixed-width data types, the raw pointers are used to point to the values to be returned. In these cases, the caller is responsible for allocating enough storage for the values to be returned.

Implements `ibis::table`.

3.17.2.11 `int64_t ibis::bord::getColumnAsStrings (const char *, std::vector< std::string > &) const` [virtual]

Retrieve the null-terminated strings as a vector of `std::string` objects.

Both `ibis::CATEGORY` and `ibis::TEXT` types can be retrieved using this function.

Implements `ibis::table`.

3.17.2.12 `int64_t ibis::bord::getColumnAsUBytes (const char *, unsigned char *) const` [virtual]

Retrieve all values of the named column.

The member functions of this class only support access to whole column at a time. Use `table::cursor` class for row-wise accesses. For fixed-width data types, the raw pointers are used to point to the values to be returned. In these cases, the caller is responsible for allocating enough storage for the values to be returned.

Implements `ibis::table`.

3.17.2.13 `int64_t ibis::bord::getColumnAsUInts (const char *, uint32_t *) const` [virtual]

Retrieve all values of the named column.

The member functions of this class only support access to whole column at a time. Use `table::cursor` class for row-wise accesses. For fixed-width data types, the raw pointers are used to point to the values to be returned. In these cases, the caller is responsible for allocating enough storage for the values to be returned.

Implements [ibis::table](#).

3.17.2.14 `int64_t ibis::bord::getColumnAsULongs (const char *, uint64_t *) const` [virtual]

Retrieve all values of the named column.

The member functions of this class only support access to whole column at a time. Use `table::cursor` class for row-wise accesses. For fixed-width data types, the raw pointers are used to point to the values to be returned. In these cases, the caller is responsible for allocating enough storage for the values to be returned.

Implements [ibis::table](#).

3.17.2.15 `int64_t ibis::bord::getColumnAsUShorts (const char *, uint16_t *) const` [virtual]

Retrieve all values of the named column.

The member functions of this class only support access to whole column at a time. Use `table::cursor` class for row-wise accesses. For fixed-width data types, the raw pointers are used to point to the values to be returned. In these cases, the caller is responsible for allocating enough storage for the values to be returned.

Implements [ibis::table](#).

3.17.2.16 `long ibis::bord::getHistogram (const char *, const char *, double, double, double, std::vector< size_t > &) const` [virtual]

Compute the histogram of the named column.

This version uses the user specified bins: `[begin, begin+stride)` `[begin+stride, begin+2*stride)`, A record is placed in bin $(x - \text{begin}) / \text{stride}$, where the first bin is bin 0. This gives a total of $(\text{end} - \text{begin}) / \text{stride}$ bins.

Note:

Records (rows) outside of the range `[begin, end]` are not counted.

Non-positive `stride` is considered as an error.

If `end` is less than `begin`, an empty array `counts` is returned along with return value 0.

Implements [ibis::table](#).

3.17.2.17 `long ibis::bord::getHistogram2D (const char *, const char *, double, double, double, const char *, double, double, double, std::vector< size_t > &) const` [virtual]

Compute a two-dimension histogram on columns `cname1` and `name2`.

The bins along each dimension are defined the same way as in function [getHistogram](#). The array `counts` stores the two-dimensional bins with the first dimension as the slow varying dimension following C convention for ordering multi-dimensional arrays.

Implements [ibis::table](#).

3.17.2.18 `long ibis::bord::getHistogram3D (const char *, const char *, double, double, double, const char *, double, double, double, const char *, double, double, double, std::vector< size_t > &) const` [virtual]

Compute a three-dimensional histogram on the named columns.

The triplets `<begin, end, stride>` are used the same ways in [getHistogram](#) and [getHistogram2D](#). The three dimensional bins are linearized in `counts` with the first being the slowest varying dimension and the third being the fastest varying dimension following the C convention for ordering multi-dimensional arrays.

Implements [ibis::table](#).

3.17.2.19 `ibis::table * ibis::bord::groupby (const ibis::table::stringList &) const` [inline, virtual]

Perform aggregate functions on the current table.

It produces a new table. The list of strings passed to this function are interpreted as a set of names followed by a set of functions. Currently, only functions COUNT, AVG, MIN, MAX, and SUM are supported.

Implements [ibis::table](#).

3.17.2.20 `virtual void ibis::bord::indexSpec (const char *, const char *)` [inline, virtual]

Replace the current indexing option.

If no column name is specified, it resets the indexing option for the table.

Implements [ibis::table](#).

3.17.2.21 `virtual const char* ibis::bord::indexSpec (const char *) const` [inline, virtual]

Retrieve the current indexing option.

If no column name is specified, it retrieve the indexing option for the table.

Implements [ibis::table](#).

3.17.2.22 `void ibis::bord::orderby (const ibis::table::stringList &)` [inline, virtual]

Reorder the rows.

Sort the rows in ascending order of the columns specified in the list of column names. This function is not designated `const` because though it does not change the content in SQL logic, but it may change internal representations.

Note:

If an empty list is passed to this function, it will reorder rows using all columns with the column having the smallest number of distinct values first.

Implements [ibis::table](#).

The documentation for this class was generated from the following files:

- [bord.h](#)
- [bord.cpp](#)

3.18 `ibis::bord::column` Class Reference

An in-memory version of [ibis::column](#).

```
#include <bord.h>
```

Public Member Functions

- `column` (const [column](#) &rhs)
- `column` (const `ibis::bord::part *`tbl, const [ibis::column](#) &, void *buf)

Note:

Transfer the ownership of st to the new column object.

- `column` (const `ibis::bord::part *`tbl, `ibis::TYPE_T` t, const char *name, void *buf, const char *desc="", double low=DBL_MAX, double high=-DBL_MAX)
- virtual void `computeMinMax` (const char *, double &min, double &max) const

- virtual void **computeMinMax** (const char *dir)
- virtual void **computeMinMax** ()
- int **dump** (std::ostream &out, size_t i) const
- virtual long **evaluateRange** (const `ibis::qContinuousRange` &cmp, const `ibis::bitvector` &mask, `ibis::bitvector` &res) const
- void * **getArray** () const
- template<typename T> int **getRawData** (`array_t`< T > &vals) const

Note:

NO type check, caller need to make sure the correct type is specified.

- virtual `ibis::fileManager::storage` * **getRawData** () const
- int **limit** (size_t nr)
- void **reverseRows** ()
- virtual `array_t`< double > * **selectDoubles** (const `ibis::bitvector` &mask) const

Put the selected values into an array as doubles.

- virtual `array_t`< float > * **selectFloats** (const `ibis::bitvector` &mask) const

Put selected values of a float column into an array.

- virtual `array_t`< int32_t > * **selectInts** (const `ibis::bitvector` &mask) const
- virtual `array_t`< int64_t > * **selectLongs** (const `ibis::bitvector` &mask) const

Can be called on all integral types.

- virtual `array_t`< uint32_t > * **selectUInts** (const `ibis::bitvector` &mask) const

Can be called on columns of unsigned integral types, UINT, CATEGORY, USHORT, and UBYTE.

Protected Member Functions

- const `column` & **operator=** (const `column` &)

Protected Attributes

- void * **buffer**

3.18.1 Detailed Description

An in-memory version of `ibis::column`.

The void* points to an `array`<T> object where the type T is designated by column type.

3.18.2 Member Function Documentation**3.18.2.1 `array_t`< double > * `ibis::bord::column::selectDoubles` (const `ibis::bitvector` & *mask*) const**
[virtual]

Put the selected values into an array as doubles.

Note:

Any column type could be selected as doubles. Other selectXXXs function only work on the same data type. This is the only function that allows one to convert to a different type. This is mainly to

3.18.2.2 `array_t< int64_t > * ibis::bord::column::selectLongs (const ibis::bitvector & mask) const` `[virtual]`

Can be called on all integral types.

Note that 64-byte unsigned integers are simply treated as signed integer. This may cause the values to be interpreted incorrectly. Shorter version of unsigned integers are treated correctly as positive values.

The documentation for this class was generated from the following files:

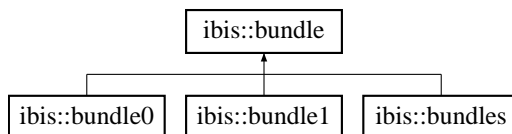
- [bord.h](#)
- [bord.cpp](#)

3.19 `ibis::bundle` Class Reference

The public interface of bundles.

```
#include <bundle.h>
```

Inheritance diagram for `ibis::bundle`:



Public Member Functions

- virtual void * `columnArray` (uint32_t j) const
Return the pointer to the underlying array used to store the `j`th column of the bundle.
- virtual `ibis::TYPE_T` `columnType` (uint32_t j) const
Return the type used to store the values of the `j`th column of the bundle.
- const `ibis::RIDSet` * `getRIDs` () const
Return the pointer to all RIDs.
- const `ibis::RIDSet` * `getRIDs` (uint32_t ind) const
Return the RIDs of the `ind`th bundle.
- uint32_t `numRowsInBundle` (uint32_t ind) const
Compute the number of rows in bundle `ind`.
- virtual void `print` (std::ostream &out) const=0
Print the bundle values to the specified output stream.
- virtual void `printAll` (std::ostream &out) const=0
Print the bundle values along with the RIDs.
- virtual void `reorder` (const char *names, int direction)=0
Re-order the bundles according to the new keys.
- uint32_t `rowCounts` (`array_t< uint32_t >` &cnt) const
Compute the number of rows in each group(bundle).

- virtual uint32_t **size** () const
Return the number of bundles.
- void **sortRIDs** (uint32_t i, uint32_t j)
- void **swapRIDs** (uint32_t i, uint32_t j)
- virtual long **truncate** (const char *names, int direction, uint32_t keep)=0
Truncate the list of bundle based on specified keys.
- virtual long **truncate** (uint32_t keep)=0
Truncate the list of bundles.
- virtual uint32_t **width** () const
Return the width of the bundles.
- virtual void **write** (const **ibis::query** &q) const=0
Write the bundle to the directory for the query q.
- virtual double **getDouble** (uint32_t, uint32_t) const
Return the maximum double value.
- virtual float **getFloat** (uint32_t, uint32_t) const
Return the maximal float value.
- virtual int32_t **getInt** (uint32_t, uint32_t) const
Retrieve a specific value.
- virtual int64_t **getLong** (uint32_t, uint32_t) const
Return the maximal int value.
- virtual std::string **getString** (uint32_t, uint32_t) const
Retrieve a string value.
- virtual uint32_t **getUInt** (uint32_t, uint32_t) const
Return the maximal unsigned int value.
- virtual uint64_t **getULong** (uint32_t, uint32_t) const
Return the maximal unsigned int value.

Static Public Member Functions

- static **bundle** * **create** (const **ibis::part** &, const **ibis::selected** &sel, const std::vector< void * > &vals)
- static **bundle** * **create** (const **ibis::query** &q, const **ibis::bitvector** &hits)
Create new bundle from a hit vector. Write info to q.dir().
- static **bundle** * **create** (const **ibis::query** &q)
Create a new bundle from previously stored information.
- static const **ibis::RIDSet** * **readRIDs** (const char *dir, const uint32_t i)
Return the RIDs related to the ith bundle.

Protected Member Functions

- `bundle` (const `ibis::query` &q, const `ibis::bitvector` &hits)
- `bundle` (const `ibis::query` &q)
- `bundle` (const `ibis::selected` &c)

Protected Attributes

- const `ibis::selected` & `comps`
- const char * `id`
- bool `infile`
- `ibis::RIDSet` * `rids`
- `array_t`< `uint32_t` > * `starts`

3.19.1 Detailed Description

The public interface of bundles.

3.19.2 Member Function Documentation

3.19.2.1 `int32_t ibis::bundle::getInt (uint32_t, uint32_t) const` [virtual]

Retrieve a specific value.

Numerical values will be casted into the return type.

Note:

Most compilers will emit numerous complains about the potential data loss due to type conversions. User should employ the correct types to avoid actual loss of precision.

Reimplemented in `ibis::bundle1`, and `ibis::bundles`.

3.19.2.2 `std::string ibis::bundle::getString (uint32_t, uint32_t) const` [virtual]

Retrieve a string value.

It converts any data type to its string representation through the string stream library.

Note:

This is generic, but slow!

Reimplemented in `ibis::bundle1`, and `ibis::bundles`.

3.19.2.3 `uint32_t ibis::bundle::rowCounts (array_t< uint32_t > & cnt) const`

Compute the number of rows in each group(bundle).

Return the number of bundles.

The documentation for this class was generated from the following files:

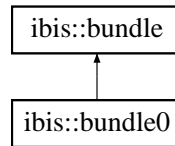
- `bundle.h`
- `bundle.cpp`

3.20 `ibis::bundle0` Class Reference

The null bundle. It contains only a list of RIDs.

```
#include <bundle.h>
```

Inheritance diagram for `ibis::bundle0`:



Public Member Functions

- **bundle0** (const [ibis::query](#) &q, const [ibis::bitvector](#) &hits)
- **bundle0** (const [ibis::query](#) &q)
- virtual void [print](#) (std::ostream &out) const
Print the bundle values to the specified output stream.
- virtual void [printAll](#) (std::ostream &out) const
Print the bundle values along with the RIDs.
- virtual void [reorder](#) (const char *names, int direction)
Re-order the bundles according to the new keys.
- virtual [uint32_t size](#) () const
Return the number of bundles.
- virtual long [truncate](#) (const char *names, int direction, [uint32_t](#) keep)
Truncate the list of bundle based on specified keys.
- virtual long [truncate](#) ([uint32_t](#) keep)
Truncate the list of bundles.
- virtual void [write](#) (const [ibis::query](#) &q) const
Write the bundle to the directory for the query q.

3.20.1 Detailed Description

The null bundle. It contains only a list of RIDs.

The documentation for this class was generated from the following files:

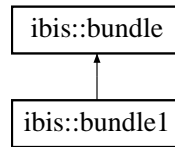
- [bundle.h](#)
- [bundle.cpp](#)

3.21 `ibis::bundle1` Class Reference

The bundle with only one component.

```
#include <bundle.h>
```

Inheritance diagram for `ibis::bundle1`:



Public Member Functions

- **bundle1** (const `ibis::part` &tbl, const `ibis::selected` &sel, const `std::vector< void * >` &vals)
- **bundle1** (const `ibis::query` &q, const `ibis::bitvector` &hits)
- **bundle1** (const `ibis::query` &q)
- virtual void * **columnArray** (uint32_t j) const
Return the pointer to the underlying array used to store the jth column of the bundle.
- virtual `ibis::TYPE_T` **columnType** (uint32_t j) const
Return the type used to store the values of the jth column of the bundle.
- virtual double **getDouble** (uint32_t, uint32_t) const
Return the maximal value defined in the class numeric_limits.
- virtual float **getFloat** (uint32_t, uint32_t) const
Return the maximal value defined in the class numeric_limits.
- virtual int32_t **getInt** (uint32_t, uint32_t) const
Return the maximal value defined in the class numeric_limits.
- virtual int64_t **getLong** (uint32_t, uint32_t) const
Return the maximal value defined in the class numeric_limits.
- virtual `std::string` **getString** (uint32_t, uint32_t) const
Convert any value to its string representation through `std::ostringstream`.
- virtual uint32_t **getUInt** (uint32_t, uint32_t) const
Return the maximal value defined in the class numeric_limits.
- virtual uint64_t **getULong** (uint32_t, uint32_t) const
Return the maximal value defined in the class numeric_limits.
- virtual void **print** (`std::ostream` &out) const
Print the bundle values to the specified output stream.
- virtual void **printAll** (`std::ostream` &out) const
Print the bundle values along with the RIDs.
- virtual void **reorder** (const char *names, int direction)
Re-order the bundles according to the new keys.
- virtual uint32_t **size** () const
Return the number of bundles.

- virtual long [truncate](#) (const char *names, int direction, uint32_t keep)
Truncate the list of bundle based on specified keys.
- virtual long [truncate](#) (uint32_t keep)
Truncate the list of bundles.
- virtual uint32_t [width](#) () const
Return the width of the bundles.
- virtual void [write](#) (const [ibis::query](#) &) const
Write the bundle to the directory for the query q.

3.21.1 Detailed Description

The bundle with only one component.

The documentation for this class was generated from the following files:

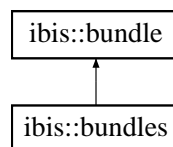
- [bundle.h](#)
- [bundle.cpp](#)

3.22 **ibis::bundles** Class Reference

The bundle with multiple components.

```
#include <bundle.h>
```

Inheritance diagram for `ibis::bundles`:



Public Member Functions

- **bundles** (const [ibis::part](#) &tbl, const [ibis::selected](#) &sel, const std::vector< void * > &vals)
- **bundles** (const [ibis::query](#) &q, const [ibis::bitvector](#) &hits)
- **bundles** (const [ibis::query](#) &q)
- virtual void * [columnArray](#) (uint32_t j) const
Return the pointer to the underlying array used to store the jth column of the bundle.
- virtual [ibis::TYPE_T](#) [columnType](#) (uint32_t j) const
Return the type used to store the values of the jth column of the bundle.
- virtual double [getDouble](#) (uint32_t, uint32_t) const
Return the maximal value defined in the class numeric_limits.
- virtual float [getFloat](#) (uint32_t, uint32_t) const
Return the maximal value defined in the class numeric_limits.

- virtual int32_t [getInt](#) (uint32_t, uint32_t) const
Return the maximal value defined in the class numeric_limits.
- virtual int64_t [getLong](#) (uint32_t, uint32_t) const
Return the maximal value defined in the class numeric_limits.
- virtual std::string [getString](#) (uint32_t, uint32_t) const
Convert any value to its string representation through std::ostringstream.
- virtual uint32_t [getUInt](#) (uint32_t, uint32_t) const
Return the maximal value defined in the class numeric_limits.
- virtual uint64_t [getULong](#) (uint32_t, uint32_t) const
Return the maximal value defined in the class numeric_limits.
- virtual void [print](#) (std::ostream &out) const
Print the bundle values to the specified output stream.
- virtual void [printAll](#) (std::ostream &out) const
Print the bundle values along with the RIDs.
- virtual void [reorder](#) (const char *names, int direction)
Reorder the bundles according to the keys (names) given.
- virtual uint32_t [size](#) () const
Return the number of bundles.
- virtual long [truncate](#) (const char *names, int direction, uint32_t keep)
Reorder the bundles according to the keys (names) given.
- virtual long [truncate](#) (uint32_t keep)
Truncate the list of bundles.
- virtual uint32_t [width](#) () const
Return the width of the bundles.
- virtual void [write](#) (const [ibis::query](#) &) const
Write the bundle to the directory for the query q.

3.22.1 Detailed Description

The bundle with multiple components.

3.22.2 Member Function Documentation

3.22.2.1 void [ibis::bundles::reorder](#) (const char * *names*, int *direction*) [virtual]

Reorder the bundles according to the keys (names) given.

turn counts back into starting positions (starts)

Implements [ibis::bundle](#).

3.22.2.2 `long ibis::bundles::truncate (const char * names, int direction, uint32_t keep)` [virtual]

Reorder the bundles according to the keys (names) given.

Keep only the first `keep` elements. If `direction < 0`, keep the largest ones, otherwise keep the smallest ones.

turn counts back into starting positions (starts)

Implements `ibis::bundle`.

The documentation for this class was generated from the following files:

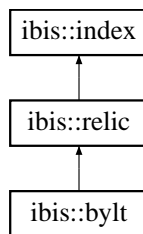
- [bundle.h](#)
- [bundle.cpp](#)

3.23 `ibis::bylt` Class Reference

The two-level range-equality code.

```
#include <irelic.h>
```

Inheritance diagram for `ibis::bylt`:



Public Member Functions

- virtual long [append](#) (const char *dt, const char *df, uint32_t nnew)
Extend the index.
- [bylt](#) (const [ibis::column](#) *c, [ibis::fileManager::storage](#) *st, uint32_t offset=8)
The leading portion of the index file is the same as `ibis::relic`, which allows the constructor of the base class to work properly.
- [bylt](#) (const [ibis::column](#) *c=0, const char *f=0)
- virtual uint32_t [estimate](#) (const [ibis::qContinuousRange](#) &expr) const
Returns an upper bound on the number of hits.
- virtual double [estimateCost](#) (const [ibis::qContinuousRange](#) &expr) const
Estimate the code of evaluate a range condition.
- virtual long [evaluate](#) (const [ibis::qContinuousRange](#) &expr, [ibis::bitvector](#) &hits) const
To evaluate the exact hits.
- virtual const char * [name](#) () const
Returns the name of the index, similar to the function `type`, but returns a string instead.
- virtual void [print](#) (std::ostream &out) const
Prints human readable information.

- virtual void `read` (`ibis::fileManager::storage *st`)
Reconstructs an index from an array of bytes.
- virtual void `read` (`const char *idxfile`)
Reconstructs an index from the named file.
- virtual `INDEX_TYPE type` () const
Returns an index type identifier.
- virtual void `write` (`const char *dt`) const
Save index to a file.

Protected Member Functions

- virtual void `clear` ()
Clear the existing content.

3.23.1 Detailed Description

The two-level range-equality code.

Note:

Bylt is Danish word for pack, the name of the binned version of the two-level range-equality code.

3.23.2 Constructor & Destructor Documentation

3.23.2.1 `ibis::bylt::bylt` (`const ibis::column *c`, `ibis::fileManager::storage *st`, `uint32_t start = 8`)

The leading portion of the index file is the same as `ibis::relic`, which allows the constructor of the base class to work properly.

The content following the last bitvector in `ibis::relic` is as follows, writeCoarse.

`nc` (`uint32_t`) – number of coarse bins. `cbounds` (`unsigned[nc+1]`) – boundaries of the coarse bins. `coffsets`(`int32_t[nc+1]`) – starting position of the coarse level bitmaps. `cbits` (`bitvector[nc]`) – bitvector laid out one after another.

3.23.3 Member Function Documentation

3.23.3.1 `long ibis::bylt::evaluate` (`const ibis::qContinuousRange & expr`, `ibis::bitvector & hits`) const [virtual]

To evaluate the exact hits.

On success, return the number of hits, otherwise a negative value is returned.

Reimplemented from `ibis::relic`.

3.23.3.2 `void ibis::bylt::print` (`std::ostream & out`) const [virtual]

Prints human readable information.

Outputs information about the index as text to the specified output stream.

Reimplemented from `ibis::relic`.

3.23.3.3 `void ibis::bylt::read (ibis::fileManager::storage * st)` [virtual]

Reconstructs an index from an array of bytes.

Intended for internal use only!

Reimplemented from [ibis::relic](#).

3.23.3.4 `void ibis::bylt::read (const char * idxfile)` [virtual]

Reconstructs an index from the named file.

The name can be the directory containing an index file. In this case, the name of the index file must be the name of the column followed by ".idx" suffix.

Reimplemented from [ibis::relic](#).

3.23.3.5 `void ibis::bylt::write (const char * dt) const` [virtual]

Save index to a file.

Outputs the index in a compact binary format to the named file or directory. The index file contains a header that can be identified by the function `isIndex`.

Reimplemented from [ibis::relic](#).

The documentation for this class was generated from the following files:

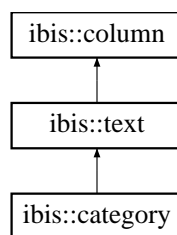
- [irelic.h](#)
- [ixbylt.cpp](#)

3.24 `ibis::category` Class Reference

A specialized low-cardinality text field.

```
#include <category.h>
```

Inheritance diagram for `ibis::category`:

**Public Member Functions**

- virtual long [append](#) (const char *dt, const char *df, const uint32_t nold, const uint32_t nnew, const uint32_t nbuf, char *buf)
Append the content in df to the directory dt.
- **category** (const [part](#) *tbl, const char *name, const char *value, const char *dir=0, uint32_t nevt=0)
- [category](#) (const [ibis::column](#) &col)
Copy constructor. Copy from a column object with KEY type.
- **category** (const [part](#) *tbl, const char *name)
- **category** (const [part](#) *tbl, FILE *file)

- virtual double **estimateCost** (const `ibis::qMultiString &cmp`) const
- virtual double **estimateCost** (const `ibis::qString &cmp`) const
- virtual const char * **getKey** (uint32_t i) const
*Return the *i*th value in the dictionary.*
- virtual const char * **isKey** (const char *str) const
Is the given string one of the keys in the dictionary? Return a null pointer if not.
- virtual void **print** (std::ostream &out) const
- virtual long **search** (const std::vector< std::string > &vals) const
Estimate the total number of matches for a list of strings.
- virtual long **search** (const char *str) const
Estimate the number of matches.
- virtual long **search** (const std::vector< std::string > &vals, `ibis::bitvector` &hits) const
Match a list of strings.
- virtual long **search** (const char *str, `ibis::bitvector` &hits) const
Match a particular string.
- virtual `array_t< uint32_t >` * **selectUInts** (const `bitvector` &mask) const
Return the integer values of the records marked 1 in the mask.
- virtual void **write** (FILE *file) const
Write the current content to the TDC file.

3.24.1 Detailed Description

A specialized low-cardinality text field.

It is also known as control values, or categorical values. This implementation directly converts string values into bitvectors (as `ibis::relic`), and does not store integer version of the string.

Note:

Value zero (0) is reserved for NULL values.

The documentation for this class was generated from the following files:

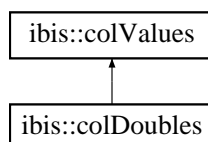
- [category.h](#)
- [category.cpp](#)

3.25 `ibis::colDoubles` Class Reference

A class to store double precision floating-point values.

```
#include <colValues.h>
```

Inheritance diagram for `ibis::colDoubles`:



Public Member Functions

- virtual void **bottomk** (uint32_t k, array_t< uint32_t > &ind) const
Return the positions of the k smallest elements.
- **colDoubles** (const **ibis::column** *c, void *vals)
- **colDoubles** (const **ibis::column** *c, **ibis::fileManager::storage** *store, const uint32_t start, const uint32_t nelm)
- **colDoubles** (const **ibis::column** *c, const **ibis::bitvector** &hits)
- virtual uint32_t **elementSize** () const
- virtual bool **empty** () const
- virtual void **erase** (uint32_t i, uint32_t j)
- virtual void * **getArray** () const
Return the pointer to the pointer to underlying array_t<T> object.
- virtual double **getDouble** (uint32_t i) const
- virtual float **getFloat** (uint32_t i) const
- virtual int32_t **getInt** (uint32_t i) const
- virtual int64_t **getLong** (uint32_t i) const
- virtual double **getMax** () const
- virtual double **getMin** () const
- virtual double **getSum** () const
- virtual **ibis::TYPE_T** **getType** () const
Return the type of the data stored.
- virtual uint32_t **getUInt** (uint32_t i) const
- virtual uint64_t **getULong** (uint32_t i) const
- virtual const **ibis::column** * **operator** → () const
Provide a pointer to the column containing the selected values.
- virtual void **reduce** (const array_t< uint32_t > &starts, **ibis::selected::FUNCTION** func)
- virtual void **reduce** (const array_t< uint32_t > &starts)
- virtual void **reorder** (const array_t< uint32_t > &ind)
Reorder the values according to the specified indices.
- virtual array_t< uint32_t > * **segment** (const array_t< uint32_t > *old=0) const
Produce an array of the starting positions of values that are the same.
- virtual uint32_t **size** () const
- virtual void **sort** (uint32_t i, uint32_t j, array_t< uint32_t > &neworder) const
Sort rows in the range [i, j).
- virtual void **sort** (uint32_t i, uint32_t j, **bundle** *bdl, **colList::iterator** head, **colList::iterator** tail)
Sort rows in the range [i, j).
- virtual void **sort** (uint32_t i, uint32_t j, **bundle** *bdl)
Sort rows in the range [i, j).
- void **swap** (**colDoubles** &rhs)
- virtual void **swap** (uint32_t i, uint32_t j)
- virtual void **topk** (uint32_t k, array_t< uint32_t > &ind) const
Return the positions of the k largest elements.
- virtual long **truncate** (uint32_t keep)

Truncate the number element to no more than keep.

- virtual void `write` (std::ostream &out, uint32_t i) const
Write ith element as text.
- virtual uint32_t `write` (FILE *fptr) const
Write out whole array as binary.

3.25.1 Detailed Description

A class to store double precision floating-point values.

3.25.2 Member Function Documentation

3.25.2.1 virtual void `ibis::colDoubles::reorder` (const `array_t`< uint32_t > & *ind*) [`inline`, `virtual`]

Reorder the values according to the specified indices.

`New[i] = Old[ind[i]].`

Implements `ibis::colValues`.

3.25.2.2 void `ibis::colDoubles::sort` (uint32_t *i*, uint32_t *j*, `array_t`< uint32_t > & *neworder*) const [`virtual`]

Sort rows in the range [*i*, *j*).

Output the new order in array *neworder*.

Implements `ibis::colValues`.

3.25.2.3 void `ibis::colDoubles::sort` (uint32_t *i*, uint32_t *j*, `bundle` * *bdl*, `colList::iterator` *head*, `colList::iterator` *tail*) [`virtual`]

Sort rows in the range [*i*, *j*).

Also sort the columns between [*head*, *tail*).

Implements `ibis::colValues`.

The documentation for this class was generated from the following files:

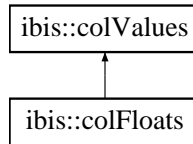
- `colValues.h`
- `colValues.cpp`

3.26 `ibis::colFloats` Class Reference

A class to store single precision float-point values.

```
#include <colValues.h>
```

Inheritance diagram for `ibis::colFloats`:



Public Member Functions

- virtual void `bottomk` (uint32_t k, array_t< uint32_t > &ind) const
Return the positions of the k smallest elements.
- `colFloats` (const `ibis::column` *c, void *vals)
- `colFloats` (const `ibis::column` *c, `ibis::fileManager::storage` *store, const uint32_t start, const uint32_t nelm)
- `colFloats` (const `ibis::column` *c, const `ibis::bitvector` &hits)
- virtual uint32_t `elementSize` () const
- virtual bool `empty` () const
- virtual void `erase` (uint32_t i, uint32_t j)
- virtual void * `getArray` () const
Return the pointer to the pointer to underlying array_t<T> object.
- virtual double `getDouble` (uint32_t i) const
- virtual float `getFloat` (uint32_t i) const
- virtual int32_t `getInt` (uint32_t i) const
- virtual int64_t `getLong` (uint32_t i) const
- virtual double `getMax` () const
- virtual double `getMin` () const
- virtual double `getSum` () const
- virtual `ibis::TYPE_T` `getType` () const
Return the type of the data stored.
- virtual uint32_t `getUInt` (uint32_t i) const
- virtual uint64_t `getULong` (uint32_t i) const
- virtual const `ibis::column` * `operator` → () const
Provide a pointer to the column containing the selected values.
- virtual void `reduce` (const array_t< uint32_t > &starts, `ibis::selected::FUNCTION` func)
- virtual void `reduce` (const array_t< uint32_t > &starts)
- virtual void `reorder` (const array_t< uint32_t > &ind)
Reorder the values according to the specified indices.
- virtual array_t< uint32_t > * `segment` (const array_t< uint32_t > *old=0) const
Produce an array of the starting positions of values that are the same.
- virtual uint32_t `size` () const
- virtual void `sort` (uint32_t i, uint32_t j, array_t< uint32_t > &neworder) const
Sort rows in the range [i, j).
- virtual void `sort` (uint32_t i, uint32_t j, `bundle` *bdl, `colList::iterator` head, `colList::iterator` tail)
Sort rows in the range [i, j).
- virtual void `sort` (uint32_t i, uint32_t j, `bundle` *bdl)
Sort rows in the range [i, j).

- void `swap` (`colFloats` &rhs)
- virtual void `swap` (uint32_t i, uint32_t j)
- virtual void `topk` (uint32_t k, `array_t`< uint32_t > &ind) const
Return the positions of the k largest elements.
- virtual long `truncate` (uint32_t keep)
Truncate the number element to no more than keep.
- virtual void `write` (std::ostream &out, uint32_t i) const
Write ith element as text.
- virtual uint32_t `write` (FILE *fp) const
Write out whole array as binary.

3.26.1 Detailed Description

A class to store single precision float-point values.

3.26.2 Member Function Documentation

3.26.2.1 virtual void `ibis::colFloats::reorder` (const `array_t`< uint32_t > &ind) [inline, virtual]

Reorder the values according to the specified indices.

New[i] = Old[ind[i]].

Implements `ibis::colValues`.

3.26.2.2 void `ibis::colFloats::sort` (uint32_t i, uint32_t j, `array_t`< uint32_t > &neworder) const [virtual]

Sort rows in the range [i, j).

Output the new order in array neworder.

Implements `ibis::colValues`.

3.26.2.3 void `ibis::colFloats::sort` (uint32_t i, uint32_t j, `bundle` * bdl, `colList::iterator` head, `colList::iterator` tail) [virtual]

Sort rows in the range [i, j).

Also sort the columns between [head, tail).

Implements `ibis::colValues`.

The documentation for this class was generated from the following files:

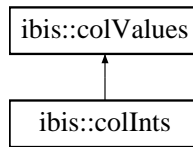
- `colValues.h`
- `colValues.cpp`

3.27 `ibis::colInts` Class Reference

A class to store integer values.

```
#include <colValues.h>
```

Inheritance diagram for `ibis::colInts`:



Public Member Functions

- virtual void **bottomk** (uint32_t k, array_t< uint32_t > &ind) const
Return the positions of the k smallest elements.
- **colInts** (const `ibis::column` *c, void *vals)
- **colInts** (const `ibis::column` *c, `ibis::fileManager::storage` *store, const uint32_t start, const uint32_t nelm)
- **colInts** (const `ibis::column` *c, const `ibis::bitvector` &hits)
- virtual uint32_t **elementSize** () const
- virtual bool **empty** () const
- virtual void **erase** (uint32_t i, uint32_t j)
- virtual void * **getArray** () const
Return the pointer to the pointer to underlying array_t<T> object.
- virtual double **getDouble** (uint32_t i) const
- virtual float **getFloat** (uint32_t i) const
- virtual int32_t **getInt** (uint32_t i) const
- virtual int64_t **getLong** (uint32_t i) const
- virtual double **getMax** () const
- virtual double **getMin** () const
- virtual double **getSum** () const
- virtual `ibis::TYPE_T` **getType** () const
Return the type of the data stored.
- virtual uint32_t **getUInt** (uint32_t i) const
- virtual uint64_t **getULong** (uint32_t i) const
- virtual const `ibis::column` * **operator** → () const
Provide a pointer to the column containing the selected values.
- virtual void **reduce** (const array_t< uint32_t > &starts, `ibis::selected::FUNCTION` func)
- virtual void **reduce** (const array_t< uint32_t > &starts)
- virtual void **reorder** (const array_t< uint32_t > &ind)
Reorder the values according to the specified indices.
- virtual array_t< uint32_t > * **segment** (const array_t< uint32_t > *old=0) const
Produce an array of the starting positions of values that are the same.
- virtual uint32_t **size** () const
- virtual void **sort** (uint32_t i, uint32_t j, array_t< uint32_t > &neworder) const
Sort rows in the range [i, j).
- virtual void **sort** (uint32_t i, uint32_t j, `bundle` *bdl, `colList::iterator` head, `colList::iterator` tail)
Sort rows in the range [i, j).

- virtual void [sort](#) (uint32_t i, uint32_t j, [bundle](#) *bdl)
Sort rows in the range [i, j).
- void [swap](#) ([colInts](#) &rhs)
- virtual void [swap](#) (uint32_t i, uint32_t j)
- virtual void [topk](#) (uint32_t k, [array_t](#)< uint32_t > &ind) const
Return the positions of the k largest elements.
- virtual long [truncate](#) (uint32_t keep)
Truncate the number element to no more than keep.
- virtual void [write](#) (std::ostream &out, uint32_t i) const
Write ith element as text.
- virtual uint32_t [write](#) (FILE *fptr) const
Write out whole array as binary.

3.27.1 Detailed Description

A class to store integer values.

3.27.2 Member Function Documentation

3.27.2.1 virtual void [ibis::colInts::reorder](#) (const [array_t](#)< uint32_t > &ind) [inline, virtual]

Reorder the values according to the specified indices.

New[i] = Old[ind[i]].

Implements [ibis::colValues](#).

3.27.2.2 void [ibis::colInts::sort](#) (uint32_t i, uint32_t j, [array_t](#)< uint32_t > &neworder) const [virtual]

Sort rows in the range [i, j).

Output the new order in array neworder.

Implements [ibis::colValues](#).

3.27.2.3 void [ibis::colInts::sort](#) (uint32_t i, uint32_t j, [bundle](#) *bdl, [colList::iterator](#) head, [colList::iterator](#) tail) [virtual]

Sort rows in the range [i, j).

Also sort the columns between [head, tail).

Implements [ibis::colValues](#).

The documentation for this class was generated from the following files:

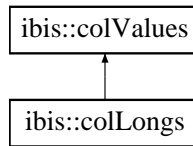
- [colValues.h](#)
- [colValues.cpp](#)

3.28 `ibis::colLongs` Class Reference

A class to store integer values.

```
#include <colValues.h>
```

Inheritance diagram for `ibis::colLongs`:



Public Member Functions

- virtual void `bottomk` (uint32_t k, array_t< uint32_t > &ind) const
Return the positions of the k smallest elements.
- `colLongs` (const `ibis::column` *c, void *vals)
- `colLongs` (const `ibis::column` *c, `ibis::fileManager::storage` *store, const uint32_t start, const uint32_t nelm)
- `colLongs` (const `ibis::column` *c, const `ibis::bitvector` &hits)
- virtual uint32_t `elementSize` () const
- virtual bool `empty` () const
- virtual void `erase` (uint32_t i, uint32_t j)
- virtual void * `getArray` () const
Return the pointer to the pointer to underlying array_t<T> object.
- virtual double `getDouble` (uint32_t i) const
- virtual float `getFloat` (uint32_t i) const
- virtual int32_t `getInt` (uint32_t i) const
- virtual int64_t `getLong` (uint32_t i) const
- virtual double `getMax` () const
- virtual double `getMin` () const
- virtual double `getSum` () const
- virtual `ibis::TYPE_T` `getType` () const
Return the type of the data stored.
- virtual uint32_t `getUInt` (uint32_t i) const
- virtual uint64_t `getULong` (uint32_t i) const
- virtual const `ibis::column` * `operator` → () const
Provide a pointer to the column containing the selected values.
- virtual void `reduce` (const array_t< uint32_t > &starts, `ibis::selected::FUNCTION` func)
- virtual void `reduce` (const array_t< uint32_t > &starts)
- virtual void `reorder` (const array_t< uint32_t > &ind)
Reorder the values according to the specified indices.
- virtual array_t< uint32_t > * `segment` (const array_t< uint32_t > *old=0) const
Produce an array of the starting positions of values that are the same.
- virtual uint32_t `size` () const
- virtual void `sort` (uint32_t i, uint32_t j, array_t< uint32_t > &neworder) const
Sort rows in the range [i, j).

- virtual void `sort` (uint32_t i, uint32_t j, `bundle` *bdl, `colList::iterator` head, `colList::iterator` tail)
Sort rows in the range [i, j).
- virtual void `sort` (uint32_t i, uint32_t j, `bundle` *bdl)
Sort rows in the range [i, j).
- void `swap` (`colLongs` &rhs)
- virtual void `swap` (uint32_t i, uint32_t j)
- virtual void `topk` (uint32_t k, `array_t`< uint32_t > &ind) const
Return the positions of the k largest elements.
- virtual long `truncate` (uint32_t keep)
Truncate the number element to no more than keep.
- virtual void `write` (std::ostream &out, uint32_t i) const
Write ith element as text.
- virtual uint32_t `write` (FILE *fptr) const
Write out whole array as binary.

3.28.1 Detailed Description

A class to store integer values.

3.28.2 Member Function Documentation

3.28.2.1 virtual void `ibis::colLongs::reorder` (const `array_t`< uint32_t > &ind) [inline, virtual]

Reorder the values according to the specified indices.

`New[i] = Old[ind[i]].`

Implements `ibis::colValues`.

3.28.2.2 void `ibis::colLongs::sort` (uint32_t i, uint32_t j, `array_t`< uint32_t > &neworder) const [virtual]

Sort rows in the range [i, j).

Output the new order in array `neworder`.

Implements `ibis::colValues`.

3.28.2.3 void `ibis::colLongs::sort` (uint32_t i, uint32_t j, `bundle` * bdl, `colList::iterator` head, `colList::iterator` tail) [virtual]

Sort rows in the range [i, j).

Also sort the columns between [head, tail).

Implements `ibis::colValues`.

The documentation for this class was generated from the following files:

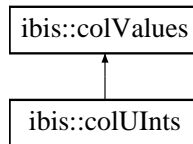
- `colValues.h`
- `colValues.cpp`

3.29 `ibis::colUints` Class Reference

A class to store unsigned integer values.

```
#include <colValues.h>
```

Inheritance diagram for `ibis::colUints`:



Public Member Functions

- virtual void `bottomk` (uint32_t k, array_t< uint32_t > &ind) const
Return the positions of the k smallest elements.
- `colUints` (const `ibis::column` *c, void *vals)
- `colUints` (const `ibis::column` *c, `ibis::fileManager::storage` *store, const uint32_t start, const uint32_t nelm)
- `colUints` (const `ibis::column` *c, const `ibis::bitvector` &hits)
- virtual uint32_t `elementSize` () const
- virtual bool `empty` () const
- virtual void `erase` (uint32_t i, uint32_t j)
- virtual void * `getArray` () const
Return the pointer to the pointer to underlying array_t<T> object.
- virtual double `getDouble` (uint32_t i) const
- virtual float `getFloat` (uint32_t i) const
- virtual int32_t `getInt` (uint32_t i) const
- virtual int64_t `getLong` (uint32_t i) const
- virtual double `getMax` () const
- virtual double `getMin` () const
- virtual double `getSum` () const
- virtual `ibis::TYPE_T` `getType` () const
Return the type of the data stored.
- virtual uint32_t `getUInt` (uint32_t i) const
- virtual uint64_t `getULong` (uint32_t i) const
- virtual const `ibis::column` * `operator` → () const
Provide a pointer to the column containing the selected values.
- virtual void `reduce` (const array_t< uint32_t > &starts, `ibis::selected::FUNCTION` func)
- virtual void `reduce` (const array_t< uint32_t > &starts)
- virtual void `reorder` (const array_t< uint32_t > &ind)
Reorder the values according to the specified indices.
- virtual array_t< uint32_t > * `segment` (const array_t< uint32_t > *old=0) const
Produce an array of the starting positions of values that are the same.
- virtual uint32_t `size` () const
- virtual void `sort` (uint32_t i, uint32_t j, array_t< uint32_t > &neworder) const
Sort rows in the range [i, j).

- virtual void **sort** (uint32_t i, uint32_t j, **bundle** *bdl, colList::iterator head, colList::iterator tail)
Sort rows in the range [i, j).
- virtual void **sort** (uint32_t i, uint32_t j, **bundle** *bdl)
Sort rows in the range [i, j).
- void **swap** (colUints &rhs)
- virtual void **swap** (uint32_t i, uint32_t j)
- virtual void **topk** (uint32_t k, **array_t**< uint32_t > &ind) const
Return the positions of the k largest elements.
- virtual long **truncate** (uint32_t keep)
Truncate the number element to no more than keep.
- virtual void **write** (std::ostream &out, uint32_t i) const
Write the ith element as text.
- virtual uint32_t **write** (FILE *fptr) const
Write out the whole array as binary.

3.29.1 Detailed Description

A class to store unsigned integer values.

3.29.2 Member Function Documentation

3.29.2.1 virtual void **ibis::colUints::reorder** (const **array_t**< uint32_t > &ind) [inline, virtual]

Reorder the values according to the specified indices.

New[i] = Old[ind[i]].

Implements [ibis::colValues](#).

3.29.2.2 void **ibis::colUints::sort** (uint32_t i, uint32_t j, **array_t**< uint32_t > &neworder) const [virtual]

Sort rows in the range [i, j).

Output the new order in array neworder.

Implements [ibis::colValues](#).

3.29.2.3 void **ibis::colUints::sort** (uint32_t i, uint32_t j, **bundle** * bdl, colList::iterator head, colList::iterator tail) [virtual]

Sort rows in the range [i, j).

Also sort the columns between [head, tail).

Implements [ibis::colValues](#).

The documentation for this class was generated from the following files:

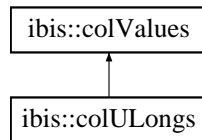
- [colValues.h](#)
- [colValues.cpp](#)

3.30 `ibis::colULongs` Class Reference

A class to store unsigned integer values.

```
#include <colValues.h>
```

Inheritance diagram for `ibis::colULongs`:



Public Member Functions

- virtual void `bottomk` (uint32_t k, array_t< uint32_t > &ind) const
Return the positions of the k smallest elements.
- `colULongs` (const `ibis::column` *c, void *vals)
- `colULongs` (const `ibis::column` *c, `ibis::fileManager::storage` *store, const uint32_t start, const uint32_t nelm)
- `colULongs` (const `ibis::column` *c, const `ibis::bitvector` &hits)
- virtual uint32_t `elementSize` () const
- virtual bool `empty` () const
- virtual void `erase` (uint32_t i, uint32_t j)
- virtual void * `getArray` () const
Return the pointer to the pointer to underlying array_t<T> object.
- virtual double `getDouble` (uint32_t i) const
- virtual float `getFloat` (uint32_t i) const
- virtual int32_t `getInt` (uint32_t i) const
- virtual int64_t `getLong` (uint32_t i) const
- virtual double `getMax` () const
- virtual double `getMin` () const
- virtual double `getSum` () const
- virtual `ibis::TYPE_T` `getType` () const
Return the type of the data stored.
- virtual uint32_t `getUInt` (uint32_t i) const
- virtual uint64_t `getULong` (uint32_t i) const
- virtual const `ibis::column` * `operator →` () const
Provide a pointer to the column containing the selected values.
- virtual void `reduce` (const array_t< uint32_t > &starts, `ibis::selected::FUNCTION` func)
- virtual void `reduce` (const array_t< uint32_t > &starts)
- virtual void `reorder` (const array_t< uint32_t > &ind)
Reorder the values according to the specified indices.
- virtual array_t< uint32_t > * `segment` (const array_t< uint32_t > *old=0) const
Produce an array of the starting positions of values that are the same.
- virtual uint32_t `size` () const
- virtual void `sort` (uint32_t i, uint32_t j, array_t< uint32_t > &neworder) const
Sort rows in the range [i, j).

- virtual void `sort` (`uint32_t i`, `uint32_t j`, `bundle *bdl`, `colList::iterator head`, `colList::iterator tail`)
Sort rows in the range [i, j).
- virtual void `sort` (`uint32_t i`, `uint32_t j`, `bundle *bdl`)
Sort rows in the range [i, j).
- void `swap` (`colULongs &rhs`)
- virtual void `swap` (`uint32_t i`, `uint32_t j`)
- virtual void `topk` (`uint32_t k`, `array_t< uint32_t > &ind`) const
Return the positions of the k largest elements.
- virtual long `truncate` (`uint32_t keep`)
Truncate the number element to no more than keep.
- virtual void `write` (`std::ostream &out`, `uint32_t i`) const
Write the ith element as text.
- virtual `uint32_t write` (`FILE *fptr`) const
Write out the whole array as binary.

3.30.1 Detailed Description

A class to store unsigned integer values.

3.30.2 Member Function Documentation

3.30.2.1 virtual void `ibis::colULongs::reorder` (`const array_t< uint32_t > &ind`) [`inline`, `virtual`]

Reorder the values according to the specified indices.

`New[i] = Old[ind[i]].`

Implements `ibis::colValues`.

3.30.2.2 void `ibis::colULongs::sort` (`uint32_t i`, `uint32_t j`, `array_t< uint32_t > &neworder`) const [`virtual`]

Sort rows in the range [i, j).

Output the new order in array `neworder`.

Implements `ibis::colValues`.

3.30.2.3 void `ibis::colULongs::sort` (`uint32_t i`, `uint32_t j`, `bundle *bdl`, `colList::iterator head`, `colList::iterator tail`) [`virtual`]

Sort rows in the range [i, j).

Also sort the columns between [head, tail).

Implements `ibis::colValues`.

The documentation for this class was generated from the following files:

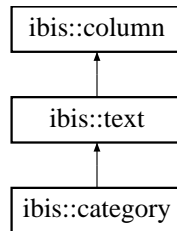
- `colValues.h`
- `colValues.cpp`

3.31 `ibis::column` Class Reference

The class to represent a column of a data table.

```
#include <column.h>
```

Inheritance diagram for `ibis::column`:



Public Member Functions

- virtual long `append` (const char *dt, const char *df, const uint32_t nold, const uint32_t nnew, const uint32_t nbuf, char *buf)

Append new data in directory df to the end of existing data in dt.
- void `binWeights` (std::vector< uint32_t > &) const
- `column` (const `column` &rhs)

copy constructor
- `column` (const `part` *tbl, `ibis::TYPE_T` t, const char *name, const char *desc="", double low=DBL_MAX, double high=-DBL_MAX)

Construct a new column of specified type.
- `column` (const `part` *tbl, FILE *file)

Reconstitute a column from the content of a file.
- virtual void `computeMinMax` (const char *dir, double &min, double &max) const

Compute the actual min/max of the data in directory dir.
- virtual void `computeMinMax` (const char *dir)
- virtual void `computeMinMax` ()

Compute the actual min/max values by actually going through all the values.
- int `contractRange` (`ibis::qContinuousRange` &rng) const
- char * `dataFileName` (const char *dir=0) const

Name of the data file in the given data directory.
- const `part` * `dataTable` () const
- void `description` (const char *d)
- const char * `description` () const
- int `elementSize` () const
- virtual double `estimateCost` (const `ibis::qMultiString` &cmp) const
- virtual double `estimateCost` (const `ibis::qString` &cmp) const
- double `estimateCost` (const `ibis::qDiscreteRange` &cmp) const
- double `estimateCost` (const `ibis::qContinuousRange` &cmp) const

Estimate the cost of evaluate the query expression.

- long `estimateRange` (const `ibis::qDiscreteRange` &cmp) const
- long `estimateRange` (const `ibis::qContinuousRange` &cmp) const
Use an index to compute an upper bound on the number of hits.
- long `estimateRange` (const `ibis::qDiscreteRange` &cmp, `ibis::bitvector` &low, `ibis::bitvector` &high) const
- long `estimateRange` (const `ibis::qContinuousRange` &cmp, `ibis::bitvector` &low, `ibis::bitvector` &high) const
Compute a lower bound and an upper bound on the number of hits using the bitmap index.
- long `evaluateRange` (const `ibis::qContinuousRange` &cmp, const `ibis::bitvector` &mask, `ibis::bitvector` &res) const
Attempt to compute the exact answer.
- int `expandRange` (`ibis::qContinuousRange` &rng) const
expand/contract range condition so that the new ranges fall exactly on the bin boundaries
- virtual const char * `findString` (const char *str) const
Determine if the input string is one of the records.
- double `getActualMax` () const
Compute the actual maximum value by reading the data or examining the index.
- double `getActualMin` () const
Compute the actual minimum value by reading the data or examining the index.
- long `getCumulativeDistribution` (std::vector< double > &bounds, std::vector< uint32_t > &counts) const
Compute the actual data distribution.
- long `getDistribution` (std::vector< double > &bbs, std::vector< uint32_t > &counts) const
Count the number of records in each bin.
- `array_t`< double > * `getDoubleArray` () const
- `array_t`< float > * `getFloatArray` () const
- `array_t`< int32_t > * `getIntArray` () const
Return all rows of the column as an `array_t` object.
- void `getNullMask` (`bitvector` &mask) const
- template<typename T> int `getRawData` (`array_t`< T > &vals) const
- `ibis::fileManager::storage` * `getRawData` () const
- virtual void `getString` (uint32_t i, std::string &val) const
- virtual const char * `getString` (uint32_t i) const
Return the internal string value for the integer.
- double `getSum` () const
Compute the sum of all values by reading the data.
- float `getUndecidable` (const `ibis::qDiscreteRange` &cmp, `ibis::bitvector` &iffy) const
- float `getUndecidable` (const `ibis::qContinuousRange` &cmp, `ibis::bitvector` &iffy) const
Compute the locations of the rows can not be decided by the index.
- void `indexSpec` (const char *spec)
- const char * `indexSpec` () const
- void `indexSpeedTest` () const
- bool `isInteger` () const

- bool **isNumeric** () const
- virtual void **loadIndex** (const char *opt=0) const throw ()
- void **logMessage** (const char *event, const char *fmt,...) const
- void **logWarning** (const char *event, const char *fmt,...) const
- void **lowerBound** (double d)
- const double & **lowerBound** () const
- const char * **name** () const
- char * **nullMaskName** () const
Name of the NULL mask file.

- uint32_t **numBins** () const
- void **preferredBounds** (std::vector< double > &) const
- virtual void **print** (std::ostream &out) const
- void **purgeIndexFile** (const char *dir=0) const
- **array_t**< char > * **selectBytes** (const **bitvector** &mask) const
*Return selected rows of the column as an **array_t** object.*

- **array_t**< double > * **selectDoubles** (const **bitvector** &mask) const
Put the selected values into an array as doubles.

- **array_t**< float > * **selectFloats** (const **bitvector** &mask) const
Put selected values of a float column into an array.

- **array_t**< int32_t > * **selectInts** (const **bitvector** &mask) const
- **array_t**< int64_t > * **selectLongs** (const **bitvector** &mask) const
Can be called on all integral types.

- **array_t**< int16_t > * **selectShorts** (const **bitvector** &mask) const
Can convert all integers 2-byte or less in length.

- virtual std::vector< std::string > * **selectStrings** (const **bitvector** &mask) const
- virtual **array_t**< uint32_t > * **selectUInts** (const **bitvector** &mask) const
Can be called on columns of unsigned integral types, UINT, CATEGORY, USHORT, and UBYTE.

- **array_t**< uint64_t > * **selectULongs** (const **bitvector** &mask) const
Can be called on all unsigned integral types.

- template<typename T> long **selectValues** (const **bitvector** &mask, **array_t**< T > &vals, **array_t**< uint32_t > &inds) const
*Select the values marked in the **bitvector** mask.*

- long **truncateData** (const char *dir, uint32_t nent, **ibis::bitvector** &mask) const
truncate the number of data entries in the named dir to nent.

- **ibis::TYPE_T** type () const
Note:
Name and type can not be changed.

- void **unloadIndex** () const
- void **upperBound** (double d)
- const double & **upperBound** () const
- virtual void **write** (FILE *file) const
Write the current content to the TDC file.

- virtual long [writeData](#) (const char *dir, uint32_t nold, uint32_t nnew, [ibis::bitvector](#) &mask, const void *val, const void *va2=0)

Record the content in array val to directory dir. Extend the mask.

Protected Member Functions

- template<typename T> void **actualMinMax** (const [array_t](#)< T > &vals, const [ibis::bitvector](#) &mask, double &min, double &max) const
- void **actualMinMax** (const char *fname, const [ibis::bitvector](#) &mask, double &min, double &max) const
Compute the actual minimum and maximum values.
- template<typename T> T **computeMax** (const [array_t](#)< T > &vals, const [ibis::bitvector](#) &mask) const
- double **computeMax** () const
- template<typename T> T **computeMin** (const [array_t](#)< T > &vals, const [ibis::bitvector](#) &mask) const
- double **computeMin** () const
- template<typename T> double **computeSum** (const [array_t](#)< T > &vals, const [ibis::bitvector](#) &mask) const
- double **computeSum** () const
- void **logError** (const char *event, const char *fmt,...) const
- long **string2int** (int fptr, [dictionary](#) &dic, uint32_t nbuf, char *buf, [array_t](#)< uint32_t > &out) const

Protected Attributes

- [ibis::index](#) * **idx**
- double **lower**
- std::string **m_bins**
- std::string **m_desc**
- std::string **m_name**
- [ibis::TYPE_T](#) **m_type**
- [ibis::bitvector](#) **mask_**
- const [part](#) * **theTable**
- double **upper**

Friends

- class **indexLock**
- class **mutexLock**
- class **writeLock**

Classes

- class [indexLock](#)
A class for controlling access of the index object of a column.
- class [info](#)
Some basic information about a column.
- class [mutexLock](#)
Provide a mutual exclusion lock on an [ibis::column](#).
- class [writeLock](#)
Provide a write lock on a [ibis::column](#) object.

3.31.1 Detailed Description

The class to represent a column of a data table.

IBIS represents user data as tables where each table consists of a number of columns. Internally, the data values for each column is stored separated from others. In relational algebra terms, this is equivalent to projecting out each attribute of a relation separately. It increases the efficiency of searching on relatively small number of attributes compared to the horizontal data organization used in typical relational database systems.

3.31.2 Constructor & Destructor Documentation

3.31.2.1 `ibis::column::column (const part * tbl, FILE * file)`

Reconstitute a column from the content of a file.

Note:

Assume the calling program has read "Begin Property/Column" already.
A well-formed column must have a valid name, i.e., ! `m_name.empty()`.

3.31.2.2 `ibis::column::column (const column & rhs)`

copy constructor

Note:

The `rwlock` can not be copied.
The index is not copied either because reference counting difficulties.

3.31.3 Member Function Documentation

3.31.3.1 `void ibis::column::actualMinMax (const char * name, const ibis::bitvector & mask, double & min, double & max) const` [`protected`]

Compute the actual minimum and maximum values.

Given a data file name, read its content to compute the actual minimum and the maximum of the data values. Only deal with four types of values, unsigned int, signed int, float and double.

3.31.3.2 `long ibis::column::append (const char * dt, const char * df, const uint32_t nold, const uint32_t nnew, const uint32_t nbuf, char * buf)` [`virtual`]

Append new data in directory `df` to the end of existing data in `dt`.

- `df` to end of file in
- `dt`.

Note:

Since this function does not compute the minimum and the maximum of the new values, it is important the minimum and the maximum is present in the corresponding `table.tdc` file. For new data without minimum and maximum, some test functions may fail.

Reimplemented in [ibis::text](#), and [ibis::category](#).

3.31.3.3 `void ibis::column::computeMinMax (const char * dir, double & min, double & max) const` [virtual]

Compute the actual min/max of the data in directory `dir`.

Report the actual min/max found back through output arguments `min` and `max`.

3.31.3.4 `void ibis::column::computeMinMax ()` [virtual]

Compute the actual min/max values by actually going through all the values.

This function reads the data in the active data directory and modifies the member variables to record the actual min/max.

3.31.3.5 `char * ibis::column::dataFileName (const char * dir = 0) const`

Name of the data file in the given data directory.

If the directory name is not given, the directory is assumed to be the current data directory of the table.

3.31.3.6 `long ibis::column::estimateRange (const ibis::qContinuousRange & cmp) const`

Use an index to compute an upper bound on the number of hits.

If no index can be computed, it will return the number of rows as the upper bound.

3.31.3.7 `long ibis::column::estimateRange (const ibis::qContinuousRange & cmp, ibis::bitvector & low, ibis::bitvector & high) const`

Compute a lower bound and an upper bound on the number of hits using the bitmap index.

If no index is available a new one will be built. If no index can be built, the lower bound will contain nothing and the the upper bound will contain everything. The two bounds are returned as bitmaps which marked the qualified rows as one, where the lower bound is stored in 'low' and the upper bound is stored in 'high'. If the bitvector 'high' has less bits than 'low', the bitvector 'low' is assumed to have an exact solution. This function always returns zero (0).

3.31.3.8 `long ibis::column::evaluateRange (const ibis::qContinuousRange & cmp, const ibis::bitvector & mask, ibis::bitvector & res) const`

Attempt to compute the exact answer.

If successful, return the number of hits, otherwise return a negative value.

3.31.3.9 `virtual const char* ibis::column::findString (const char * str) const` [inline, virtual]

Determine if the input string is one of the records.

If yes, return the pointer to the incoming string, otherwise return nil.

Reimplemented in [ibis::text](#).

3.31.3.10 `double ibis::column::getActualMax () const`

Compute the actual maximum value by reading the data or examining the index.

It returns `-DBL_MAX` in case of error.

3.31.3.11 `double ibis::column::getActualMin () const`

Compute the actual minimum value by reading the data or examining the index.

It returns `DBL_MAX` in case of error.

3.31.3.12 `long ibis::column::getCumulativeDistribution (std::vector< double > & bounds, std::vector< uint32_t > & counts) const`

Compute the actual data distribution.

It will generate an index for the column if one is not already available. The value in `cts[i]` is the number of values less than `bds[i]`. If there is no NULL values in the column, the array `cts` will start with 0 and end the number of rows in the data. The array `bds` will end with a value that is greater than the actual maximum value.

3.31.3.13 `long ibis::column::getDistribution (std::vector< double > & bbs, std::vector< uint32_t > & counts) const`

Count the number of records in each bin.

The array `bins` contains bin boundaries that defines the following bins: (... , `bins[0]`) [`bins[0]`, `bins[1]`] ... [`bins.back()`, ...). Because of the two open bins at the end, `N` bin boundaries defines `N+1` bins. The array `counts` has one more element than `bins`. This function returns the number of bins. If this function was executed successfully, the return value should be the same as the size of array `counts`, and one larger than the size of array `bbs`.

3.31.3.14 `array_t< int32_t > * ibis::column::getIntArray () const`

Return all rows of the column as an `array_t` object.

Caller is responsible for deleting the returned object.

3.31.3.15 `virtual const char* ibis::column::getString (uint32_t i) const [inline, virtual]`

Return the internal string value for the integer.

Only valid for `ibis::text` and `ibis::category`. `ibis::category` `ibis::text`

3.31.3.16 `float ibis::column::getUndecidable (const ibis::qContinuousRange & cmp, ibis::bitvector & iffy) const`

Compute the locations of the rows can not be decided by the index.

Returns the fraction of rows might satisfy the specified range condition.

3.31.3.17 `array_t< char > * ibis::column::selectBytes (const bitvector & mask) const`

Return selected rows of the column as an `array_t` object.

Caller is responsible for deleting the returned object.

3.31.3.18 `array_t< double > * ibis::column::selectDoubles (const bitvector & mask) const`

Put the selected values into an array as doubles.

Note:

Any column type could be selected as doubles. Other `selectXXXs` function only work on the same data type. This is the only function that allows one to convert to a different type. This is mainly to

3.31.3.19 `array_t< int64_t > * ibis::column::selectLongs (const bitvector & mask) const`

Can be called on all integral types.

Note that 64-byte unsigned integers are simply treated as signed integer. This may cause the values to be interpreted incorrectly. Shorter version of unsigned integers are treated correctly as positive values.

3.31.3.20 `array_t< int16_t > * ibis::column::selectShorts (const bitvector & mask) const`

Can convert all integers 2-byte or less in length.

Note that unsigned integers are simply treated as signed integers. Shorter types of signed integers are treated correctly as positive values.

3.31.3.21 `template<typename T> long ibis::column::selectValues (const bitvector & mask, array_t< T > & vals, array_t< uint32_t > & inds) const`

Select the values marked in the `bitvector` `mask`.

Select all values marked 1 in the `mask` and pack them into the output array `vals` and fill the array `inds` with the positions of the values selected. On a successful execution, it returns the number of values selected. If it returns zero (0), the contents of `vals` and `inds` are not modified. If it returns a negative number, the contents of arrays `vals` and `inds` are not guaranteed to be in particular state.

3.31.3.22 `long ibis::column::truncateData (const char * dir, uint32_t nent, ibis::bitvector & mask) const`

truncate the number of data entries in the named `dir` to `nent`.

Adjust the null mask accordingly.

The documentation for this class was generated from the following files:

- [column.h](#)
- [column.cpp](#)

3.32 `ibis::column::indexLock` Class Reference

A class for controlling access of the index object of a column.

```
#include <column.h>
```

Public Member Functions

- const `ibis::index` * `getIndex` () const
- `indexLock` (const `ibis::column` *`col`, const char *`m`)

3.32.1 Detailed Description

A class for controlling access of the index object of a column.

It directly accesses two member variables of `ibis::column` class, `idx` and `idxcnt`.

The documentation for this class was generated from the following file:

- [column.h](#)

3.33 `ibis::column::info` Class Reference

Some basic information about a column.

```
#include <column.h>
```

Public Member Functions

- `info` (const `ibis::column` &`col`)

Public Attributes

- const char * `description`
A description about the column.
- const double `expectedMax`
The expected upper bound.
- const double `expectedMin`
The expected lower bound.
- const char * `name`
Column name.
- const `ibis::TYPE_T` type
The type of the values.

3.33.1 Detailed Description

Some basic information about a column.

Can only be used if the original column used to generate the info object exists in memory.

The documentation for this class was generated from the following file:

- [column.h](#)

3.34 `ibis::column::mutexLock` Class Reference

Provide a mutual exclusion lock on an `ibis::column`.

```
#include <column.h>
```

Public Member Functions

- `mutexLock` (const `ibis::column` *col, const char *m)

3.34.1 Detailed Description

Provide a mutual exclusion lock on an `ibis::column`.

The documentation for this class was generated from the following file:

- [column.h](#)

3.35 `ibis::column::writeLock` Class Reference

Provide a write lock on a `ibis::column` object.

```
#include <column.h>
```

Public Member Functions

- `writeLock` (const `ibis::column` *col, const char *m)

3.35.1 Detailed Description

Provide a write lock on a `ibis::column` object.

The documentation for this class was generated from the following file:

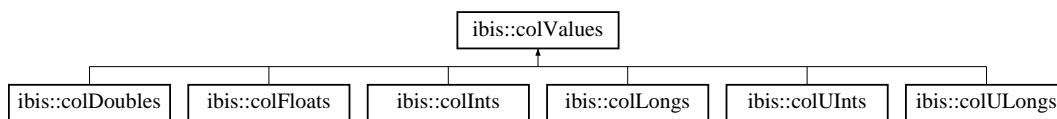
- [column.h](#)

3.36 `ibis::colValues` Class Reference

A pure virtual base class.

```
#include <colValues.h>
```

Inheritance diagram for `ibis::colValues`:



Public Member Functions

- virtual void **bottomk** (uint32_t k, array_t< uint32_t > &ind) const =0
Return the positions of the k smallest elements.
- bool **canSort** () const
- virtual uint32_t **elementSize** () const=0
- virtual bool **empty** () const=0
- virtual void **erase** (uint32_t i, uint32_t j)=0
- virtual void * **getArray** () const=0
Return the pointer to the pointer to underlying array_t<T> object.
- virtual double **getDouble** (uint32_t) const =0
- virtual float **getFloat** (uint32_t) const =0
- virtual int32_t **getInt** (uint32_t) const =0
- virtual int64_t **getLong** (uint32_t) const =0
- virtual double **getMax** () const=0
- virtual double **getMin** () const=0
- virtual double **getSum** () const=0
- virtual `ibis::TYPE_T` **getType** () const=0
Return the type of the data stored.
- virtual uint32_t **getUInt** (uint32_t) const =0
- virtual uint64_t **getULong** (uint32_t) const =0
- virtual const `ibis::column` * **operator** → () const=0
Provide a pointer to the column containing the selected values.
- virtual void **reduce** (const array_t< uint32_t > &starts, `ibis::selected::FUNCTION` func)=0
- virtual void **reduce** (const array_t< uint32_t > &starts)=0
- virtual void **reorder** (const array_t< uint32_t > &ind)=0
Reorder the values according to the specified indices.
- virtual array_t< uint32_t > * **segment** (const array_t< uint32_t > *old=0) const=0

Produce an array of the starting positions of values that are the same.

- virtual `uint32_t size () const=0`
- virtual `void sort (uint32_t i, uint32_t j, array_t< uint32_t > &neworder) const =0`
Sort rows in the range [i, j).
- virtual `void sort (uint32_t i, uint32_t j, bundle *bdl, colList::iterator head, colList::iterator tail)=0`
Sort rows in the range [i, j).
- virtual `void sort (uint32_t i, uint32_t j, bundle *bdl)=0`
Sort rows in the range [i, j).
- void `swap (colValues &rhs)`
- virtual `void swap (uint32_t i, uint32_t j)=0`
- virtual `void topk (uint32_t k, array_t< uint32_t > &ind) const =0`
Return the positions of the k largest elements.
- virtual `long truncate (uint32_t keep)=0`
Truncate the number element to no more than keep.
- virtual `void write (std::ostream &out, uint32_t i) const=0`
Write ith element as text.
- virtual `uint32_t write (FILE *fptr) const=0`
Write out whole array as binary.

Static Public Member Functions

- static `colValues * create (const ibis::column *c, void *vals)`
Construct from content of an array_t.
- static `colValues * create (const ibis::column *c, ibis::fileManager::storage *store, const uint32_t start, const uint32_t nelm)`
Construct from content of the file (pointed by store).
- static `colValues * create (const ibis::column *c, const ibis::bitvector &hits)`
Construct from a hit vector.

Protected Member Functions

- `colValues (const ibis::column *c)`

Protected Attributes

- const `ibis::column * col`
The column where the value is from.

3.36.1 Detailed Description

A pure virtual base class.

3.36.2 Member Function Documentation

3.36.2.1 `virtual void ibis::colValues::reorder (const array_t< uint32_t > & ind)` [pure virtual]

Reorder the values according to the specified indices.

`New[i] = Old[ind[i]].`

Implemented in `ibis::colInts`, `ibis::colUInts`, `ibis::colLongs`, `ibis::colULongs`, `ibis::colFloats`, and `ibis::colDoubles`.

3.36.2.2 `virtual void ibis::colValues::sort (uint32_t i, uint32_t j, array_t< uint32_t > & neworder) const` [pure virtual]

Sort rows in the range [`i`, `j`).

Output the new order in array `neworder`.

Implemented in `ibis::colInts`, `ibis::colUInts`, `ibis::colLongs`, `ibis::colULongs`, `ibis::colFloats`, and `ibis::colDoubles`.

3.36.2.3 `virtual void ibis::colValues::sort (uint32_t i, uint32_t j, bundle * bdl, colList::iterator head, colList::iterator tail)` [pure virtual]

Sort rows in the range [`i`, `j`).

Also sort the columns between [`head`, `tail`).

Implemented in `ibis::colInts`, `ibis::colUInts`, `ibis::colLongs`, `ibis::colULongs`, `ibis::colFloats`, and `ibis::colDoubles`.

The documentation for this class was generated from the following files:

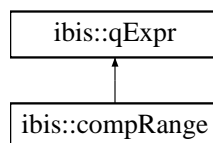
- [colValues.h](#)
- [colValues.cpp](#)

3.37 `ibis::compRange` Class Reference

The class `compRange` stores computed ranges.

```
#include <qExpr.h>
```

Inheritance diagram for `ibis::compRange`:



Public Types

- enum `OPERADOR` {
`UNKNOWN = 0`, `BITOR`, `BITAND`, `PLUS`,
`MINUS`, `MULTIPLY`, `DIVIDE`, `NEGATE`,
`POWER` }
- enum `STDFUN1` {
`ACOS = 0`, `ASIN`, `ATAN`, `CEIL`,
`COS`, `COSH`, `EXP`, `FABS`,
`FLOOR`, `FREXP`, `LOG10`, `LOG`,

`MODF`, `SIN`, `SINH`, `SQRT`,
`TAN`, `TANH` }
 • enum `STDFUN2` { `ATAN2 = 0`, `FMOD`, `LDEXP`, `POW` }
 • enum `TERM_TYPE` {
 `UNDEFINED`, `VARIABLE`, `NUMBER`, `STRING`,
 `OPERATOR`, `STDFUNCTION1`, `STDFUNCTION2`, `CUSTOMFUNCTION1`,
 `CUSTOMFUNCTION2` }

Public Member Functions

- `compRange` (const `compRange` &rhs)
- `compRange` (ibis::compRange::term *me1, `ibis::qExpr::COMPARE` lop, ibis::compRange::term *me2, `ibis::qExpr::COMPARE` rop, ibis::compRange::term *me3)
- `compRange` (ibis::compRange::term *me1, `COMPARE` lop, ibis::compRange::term *me2)
- virtual `qExpr` * `dup` () const
 Duplicate this object and return a pointer to the new copy.
- const term * `getTerm3` () const
- term * `getTerm3` ()
- bool `inRange` () const
 Evaluate the logical expression.
- virtual bool `isSimple` () const
 Is the expression simple, i.e., containing only simple range conditions joined with logical operators ?
- bool `isSimpleRange` () const
 Is this a simple range expression that can be stored as `ibis::qRange`?
- `ibis::qExpr::COMPARE` `leftOperator` () const
- bool `maybeStringCompare` () const
 Is the string a possible simple string comparison.
- virtual void `print` (std::ostream &) const
 Print the query expression.
- `ibis::qExpr::COMPARE` `rightOperator` () const
- void `setTerm3` (term *t)
- `ibis::qContinuousRange` * `simpleRange` () const

Static Public Attributes

- static char * `operator_name` []
- static char * `stdfun1_name` []
- static char * `stdfun2_name` []

Classes

- class `barrel`
 A barrel to hold a list of variables.
- class `bediener`

- class `literal`
- class `number`
- class `stdFunction1`
- class `stdFunction2`
- class `term`
- class `variable`

3.37.1 Detailed Description

The class `compRange` stores computed ranges.

It is for those comparisons involving nontrivial arithmetic expression.

The documentation for this class was generated from the following files:

- `qExpr.h`
- `qExpr.cpp`

3.38 `ibis::compRange::barrel` Class Reference

A barrel to hold a list of variables.

```
#include <qExpr.h>
```

Public Member Functions

- `barrel` (const term *const t)
- bool `equivalent` (const `barrel` &rhs) const
Is the given `barrel` of variables equivalent to this one?
- const char * `name` (uint32_t i) const
- uint32_t `recordVariable` (const char *name)
Record the specified name.
- void `recordVariable` (const term *const t)
Record the variable names appear in the `term`.
- uint32_t `size` () const
- double & `value` (uint32_t i)
- const double & `value` (uint32_t i) const

Protected Types

- typedef std::map< const char *, uint32_t, `ibis::lessi` > `termMap`

Protected Member Functions

- double `getValue` (const char *nm) const
Return the value of the named variable.
- double `getVariable` (uint32_t i) const

Protected Attributes

- `std::vector< const char * >` `namelist`
List of variable names.
- `termMap` `varmap`
Associate a variable name with a position in `varvalues` and `namelist`.
- `std::vector< double >` `varvalues`
All values are casted to double.

Friends

- class `variable`

3.38.1 Detailed Description

A barrel to hold a list of variables.

3.38.2 Member Function Documentation

3.38.2.1 `uint32_t ibis::compRange::barrel::recordVariable (const char * name)` [inline]

Record the specified name.

Return the variable number that is to be used later in functions `name` and `value` for retrieving the variable name and its value.

The documentation for this class was generated from the following files:

- `qExpr.h`
- `qExpr.cpp`

3.39 `ibis::dictionary` Class Reference

Provide a mapping between strings and integers.

```
#include <category.h>
```

Public Types

- `typedef std::map< const char *, uint32_t, ibis::lessi >` `wordList`

Public Member Functions

- void `clear` ()
- `dictionary` (const `dictionary` &dic)
- const char * `find` (const char *str) const
Is the given string in the dictionary? Return a null pointer if not.
- `uint32_t insert` (const char *str)
Insert a string if not in dictionary.

- `uint32_t insertRaw (char *str)`
Insert a string if not in dictionary.
- `uint32_t operator[] (const char *str) const`
Return an integer corresponding to the string.
- `const char * operator[] (uint32_t i) const`
Return a string corresponding to the integer.
- `void read (const char *name)`
- `uint32_t size () const`
Return the number of valid (not null) strings in the dictionary.
- `void write (const char *name) const`

Protected Member Functions

- `void copy (const dictionary &rhs)`

3.39.1 Detailed Description

Provide a mapping between strings and integers.

A utility class used by `ibis::category`. The NULL string is always the 0th string.

3.39.2 Member Function Documentation

3.39.2.1 `const char * ibis::dictionary::find (const char * str) const` `[inline]`

Is the given string in the dictionary? Return a null pointer if not.

Otherwise it returns null pointer. This function makes a little easier to determine whether a string is in a dictionary.

3.39.2.2 `uint32_t ibis::dictionary::insertRaw (char * str)` `[inline]`

Insert a string if not in dictionary.

Do not make a copy of the input string. Caller needs to check whether it is a new word in the dictionary. If it is not a new word in the dictionary, the dictionary does not take ownership of the string argument.

The documentation for this class was generated from the following files:

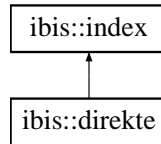
- `category.h`
- `category.cpp`

3.40 **ibis::direkte** Class Reference

Directly use the integer values as bin number to avoid some intermediate steps.

```
#include <idirekte.h>
```

Inheritance diagram for `ibis::direkte`:



Public Member Functions

- virtual long [append](#) (const char *dt, const char *df, uint32_t nnew)
Extend the index.
- virtual void [binBoundaries](#) (std::vector< double > &) const
The function binBoundaries and binWeights return bin boundaries and counts of each bin respectively.
- virtual void [binWeights](#) (std::vector< uint32_t > &) const
- [direkte](#) (const [ibis::column](#) *c, [ibis::fileManager::storage](#) *st, uint32_t offset=8)
- [direkte](#) (const [ibis::column](#) *c, const char *f=0)
Constructing a new [ibis::direkte](#) object from base data in a file.
- virtual uint32_t [estimate](#) (const [ibis::qDiscreteRange](#) &expr) const
- virtual void [estimate](#) (const [ibis::qDiscreteRange](#) &expr, [ibis::bitvector](#) &lower, [ibis::bitvector](#) &upper) const
Estimate the hits for discrete ranges, i.e., those translated from 'a IN (x, y, .
- virtual uint32_t [estimate](#) (const [ibis::qContinuousRange](#) &expr) const
Returns an upper bound on the number of hits.
- virtual void [estimate](#) (const [ibis::qContinuousRange](#) &expr, [ibis::bitvector](#) &lower, [ibis::bitvector](#) &upper) const
Computes an approximation of hits as a pair of lower and upper bounds.
- virtual double [estimateCost](#) (const [ibis::qDiscreteRange](#) &expr) const
- virtual double [estimateCost](#) (const [ibis::qContinuousRange](#) &expr) const
Estimate the code of evaluate a range condition.
- virtual long [evaluate](#) (const [ibis::qDiscreteRange](#) &expr, [ibis::bitvector](#) &hits) const
- virtual long [evaluate](#) (const [ibis::qContinuousRange](#) &expr, [ibis::bitvector](#) &hits) const
To evaluate the exact hits.
- virtual long [getCumulativeDistribution](#) (std::vector< double > &bds, std::vector< uint32_t > &cts) const
Cumulative distribution of the data.
- virtual long [getDistribution](#) (std::vector< double > &bbs, std::vector< uint32_t > &cts) const
Binned distribution of the data.
- virtual double [getMax](#) () const
The maximum value recorded in the index.
- virtual double [getMin](#) () const
The minimum value recorded in the index.
- virtual double [getSum](#) () const
Compute the approximate sum of all the values indexed.

- virtual const char * **name** () const
Returns the name of the index, similar to the function `type`, but returns a string instead.
- virtual void **print** (std::ostream &out) const
Prints human readable information.
- virtual void **read** (ibis::fileManager::storage *st)
Reconstructs an index from an array of bytes.
- virtual void **read** (const char *name)
Reconstructs an index from the named file.
- virtual void **speedTest** (std::ostream &out) const
Time some logical operations and print out their speed.
- virtual INDEX_TYPE **type** () const
Returns an index type identifier.
- virtual float **undecidable** (const ibis::qDiscreteRange &expr, ibis::bitvector &iffy) const
- virtual float **undecidable** (const ibis::qContinuousRange &expr, ibis::bitvector &iffy) const
Mark the position of the rows that can not be decided with this index.
- virtual void **write** (const char *name) const
Write the direct bitmap index to a file.

Protected Member Functions

- template<typename T> int **construct** (const char *f)
- **direkte** (const **direkte** &)
- void **locate** (const ibis::qContinuousRange &expr, uint32_t &hit0, uint32_t &hit1) const
- const **direkte** & **operator=** (const **direkte** &)

3.40.1 Detailed Description

Directly use the integer values as bin number to avoid some intermediate steps.

3.40.2 Member Function Documentation

3.40.2.1 void ibis::direkte::estimate (const ibis::qDiscreteRange & expr, ibis::bitvector & lower, ibis::bitvector & upper) const [virtual]

Estimate the hits for discrete ranges, i.e., those translated from 'a IN (x, y, . . .)'.
)'.

Reimplemented from [ibis::index](#).

3.40.2.2 `void ibis::direkte::estimate (const ibis::qContinuousRange & expr, ibis::bitvector & lower, ibis::bitvector & upper) const` [virtual]

Computes an approximation of hits as a pair of lower and upper bounds.

Parameters:

expr the query expression to be evaluated.

lower a bitvector marking a subset of the hits. All rows marked with one (1) are definitely hits.

upper a bitvector marking a superset of the hits. All hits are marked with one, but some of the rows marked one may not be hits. If the variable *upper* is empty, the variable *lower* is assumed to contain the exact answer.

Implements [ibis::index](#).

3.40.2.3 `long ibis::direkte::evaluate (const ibis::qContinuousRange & expr, ibis::bitvector & hits) const` [virtual]

To evaluate the exact hits.

On success, return the number of hits, otherwise a negative value is returned.

Implements [ibis::index](#).

3.40.2.4 `double ibis::direkte::getSum () const` [virtual]

Compute the approximate sum of all the values indexed.

If it decides that computing the sum directly from the vertical partition is more efficient, it will return NaN immediately.

Implements [ibis::index](#).

3.40.2.5 `void ibis::direkte::print (std::ostream & out) const` [virtual]

Prints human readable information.

Outputs information about the index as text to the specified output stream.

Implements [ibis::index](#).

3.40.2.6 `void ibis::direkte::read (ibis::fileManager::storage * st)` [virtual]

Reconstructs an index from an array of bytes.

Intended for internal use only!

Implements [ibis::index](#).

3.40.2.7 `void ibis::direkte::read (const char * name)` [virtual]

Reconstructs an index from the named file.

The name can be the directory containing an index file. In this case, the name of the index file must be the name of the column followed by ".idx" suffix.

Implements [ibis::index](#).

3.40.2.8 `virtual float ibis::direkte::undecidable (const ibis::qContinuousRange & expr, ibis::bitvector & iffy) const` [inline, virtual]

Mark the position of the rows that can not be decided with this index.

Parameters:

expr the range conditions to be evaluated.

iffy the bitvector marking the positions of rows that can not be decided using the index. Return value is the expected fraction of undecided rows that might satisfy the range conditions.

Implements [ibis::index](#).

The documentation for this class was generated from the following files:

- [idirekte.h](#)
- [idirekte.cpp](#)

3.41 `ibis::discretePoisson` Class Reference

Discrete random number with Poisson distribution $\exp(-x/\lambda)$.

```
#include <twister.h>
```

Public Member Functions

- **discretePoisson** ([ibis::uniformRandomNumber](#) *ur, const double lam=1.0, long m=0)
- long **next** ()
- long **operator**() ()

3.41.1 Detailed Description

Discrete random number with Poisson distribution $\exp(-x/\lambda)$.

Use the rejection-inversion algorithm of W. Hormann and G. Derflinger.

The documentation for this class was generated from the following file:

- [twister.h](#)

3.42 `ibis::discretePoisson1` Class Reference

Specialized version of the Poisson distribution $\exp(-x)$.

```
#include <twister.h>
```

Public Member Functions

- **discretePoisson1** ([ibis::uniformRandomNumber](#) *ur)
- long **next** ()
- long **operator**() ()

3.42.1 Detailed Description

Specialized version of the Poisson distribution $\exp(-x)$.

The documentation for this class was generated from the following file:

- [twister.h](#)

3.43 `ibis::discreteZipf` Class Reference

Discrete Zipf distribution: $p(k)$ is proportional to $(v+k)^{-a}$ where $a > 1$, $k \geq 0$.

```
#include <twister.h>
```

Public Member Functions

- **`discreteZipf`** (`ibis::uniformRandomNumber` *ur, double a=2.0, unsigned long v=1, unsigned long imax=ULONG_MAX)
- unsigned long **`next`** ()
- unsigned long **`operator()`** ()

3.43.1 Detailed Description

Discrete Zipf distribution: $p(k)$ is proportional to $(v+k)^{-a}$ where $a > 1$, $k \geq 0$.

It uses the rejection-inversion algorithm of W. Hormann and G. Derflinger. The values generated are in the range of $[0, imax]$ (inclusive, both ends are included).

The documentation for this class was generated from the following file:

- [twister.h](#)

3.44 `ibis::discreteZipf1` Class Reference

A specialized case of the Zipf distribution $f(x) = 1/(1+x)$.

```
#include <twister.h>
```

Public Member Functions

- **`discreteZipf1`** (`ibis::uniformRandomNumber` *ur, unsigned long imax=100)
- unsigned long **`next`** ()
- unsigned long **`operator()`** ()

3.44.1 Detailed Description

A specialized case of the Zipf distribution $f(x) = 1/(1+x)$.

The general case `discreteZipf` requires $a > 1$.

The documentation for this class was generated from the following file:

- [twister.h](#)

3.45 `ibis::discreteZipf2` Class Reference

A specialized version of the Zipf distribution $f(x) = 1/(1+x)^2$.

```
#include <twister.h>
```

Public Member Functions

- **discreteZipf2** (`ibis::uniformRandomNumber *ur`, unsigned long `imax=ULONG_MAX`)
- unsigned long **next** ()
- unsigned long **operator()** ()

Return a discrete random number in the range of [0, imax].

3.45.1 Detailed Description

A specialized version of the Zipf distribution $f(x) = 1/(1+x)^2$.

Should be much faster than using `discreteZipf(2,1,imax)`.

The documentation for this class was generated from the following file:

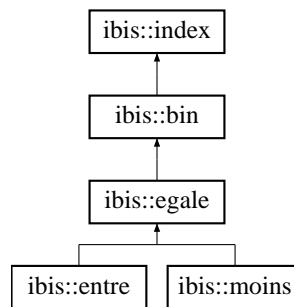
- [twister.h](#)

3.46 `ibis::egale` Class Reference

The multicomponent equality code on bins.

```
#include <ibin.h>
```

Inheritance diagram for `ibis::egale`:



Public Member Functions

- long **append** (const `array_t< uint32_t >` &ind)
Append a list of integers representing bin numbers.
- long **append** (const `ibis::egale` &tail)
- virtual long **append** (const char *dt, const char *df, uint32_t nnew)
Extend the index.
- virtual void **binBoundaries** (std::vector< double > &b) const
The function binBoundaries and binWeights return bin boundaries and counts of each bin respectively.
- virtual void **binWeights** (std::vector< uint32_t > &b) const
- **egale** (const `ibis::bin` &rhs, const uint32_t nbase=2)
- **egale** (const `ibis::column` *c, `ibis::fileManager::storage` *st, uint32_t offset=8)
- **egale** (const `ibis::column` *c=0, const char *f=0, const uint32_t nbase=2)
- virtual uint32_t **estimate** (const `ibis::qContinuousRange` &expr) const
Returns an upper bound on the number of hits.

- virtual void **estimate** (const [ibis::qContinuousRange](#) &expr, [ibis::bitvector](#) &lower, [ibis::bitvector](#) &upper) const
Computes an approximation of hits as a pair of lower and upper bounds.
- virtual long **evaluate** (const [ibis::qContinuousRange](#) &expr, [ibis::bitvector](#) &hits) const
To evaluate the exact hits.
- virtual double **getSum** () const
Compute the approximate sum of all the values indexed.
- virtual const char * **name** () const
Returns the name of the index, similar to the function `type`, but returns a string instead.
- virtual void **print** (std::ostream &out) const
Prints human readable information.
- virtual void **read** ([ibis::fileManager::storage](#) *st)
Reconstructs an index from an array of bytes.
- virtual void **read** (const char *idxfile)
Reconstructs an index from the named file.
- virtual void **speedTest** (std::ostream &out) const
Time some logical operations and print out their speed.
- virtual INDEX_TYPE **type** () const
Returns an index type identifier.
- virtual float **undecidable** (const [ibis::qContinuousRange](#) &expr, [ibis::bitvector](#) &iffy) const
Mark the position of the rows that can not be decided with this index.
- virtual void **write** (const char *dt) const
Save index to a file.

Protected Member Functions

- void **addBins_** (uint32_t ib, uint32_t ie, [ibis::bitvector](#) &res) const
- virtual void **clear** ()
Clear the existing content.
- virtual double **computeSum** () const
- void **construct** (const char *f)
- **egale** (const [ibis::column](#) *c, const char *f, const [array_t](#)< double > &bd, const [array_t](#)< uint32_t > bs)
- void **write** (int fdes) const

Protected Attributes

- [array_t](#)< uint32_t > **bases**
- [array_t](#)< uint32_t > **cnts**
- uint32_t **nbases**
- uint32_t **nbits**

3.46.1 Detailed Description

The multicomponent equality code on bins.

The word `egale` is a French word for 'equal'.

3.46.2 Member Function Documentation

3.46.2.1 `void ibis::egale::estimate (const ibis::qContinuousRange & expr, ibis::bitvector & lower, ibis::bitvector & upper) const` [virtual]

Computes an approximation of hits as a pair of lower and upper bounds.

Parameters:

expr the query expression to be evaluated.

lower a bitvector marking a subset of the hits. All rows marked with one (1) are definitely hits.

upper a bitvector marking a superset of the hits. All hits are marked with one, but some of the rows marked one may not be hits. If the variable `upper` is empty, the variable `lower` is assumed to contain the exact answer.

Reimplemented from [ibis::bin](#).

Reimplemented in [ibis::moins](#), and [ibis::entre](#).

3.46.2.2 `long ibis::egale::evaluate (const ibis::qContinuousRange & expr, ibis::bitvector & hits) const` [virtual]

To evaluate the exact hits.

On success, return the number of hits, otherwise a negative value is returned.

Reimplemented from [ibis::bin](#).

Reimplemented in [ibis::moins](#), and [ibis::entre](#).

3.46.2.3 `double ibis::egale::getSum () const` [virtual]

Compute the approximate sum of all the values indexed.

If it decides that computing the sum directly from the vertical partition is more efficient, it will return NaN immediately.

Reimplemented from [ibis::bin](#).

Reimplemented in [ibis::moins](#), and [ibis::entre](#).

3.46.2.4 `void ibis::egale::print (std::ostream & out) const` [virtual]

Prints human readable information.

Outputs information about the index as text to the specified output stream.

Reimplemented from [ibis::bin](#).

Reimplemented in [ibis::moins](#), and [ibis::entre](#).

3.46.2.5 `void ibis::egale::read (ibis::fileManager::storage * st)` [virtual]

Reconstructs an index from an array of bytes.

Intended for internal use only!

Reimplemented from [ibis::bin](#).

3.46.2.6 `void ibis::egale::read (const char * idxfile) [virtual]`

Reconstructs an index from the named file.

The name can be the directory containing an index file. In this case, the name of the index file must be the name of the column followed by ".idx" suffix.

Reimplemented from [ibis::bin](#).

3.46.2.7 `float ibis::egale::undecidable (const ibis::qContinuousRange & expr, ibis::bitvector & iffy) const [virtual]`

Mark the position of the rows that can not be decided with this index.

Parameters:

expr the range conditions to be evaluated.

iffy the bitvector marking the positions of rows that can not be decided using the index. Return value is the expected fraction of undecided rows that might satisfy the range conditions.

Reimplemented from [ibis::bin](#).

3.46.2.8 `void ibis::egale::write (const char * dt) const [virtual]`

Save index to a file.

Outputs the index in a compact binary format to the named file or directory. The index file contains a header that can be identified by the function `isIndex`.

Reimplemented from [ibis::bin](#).

Reimplemented in [ibis::moins](#), and [ibis::entre](#).

The documentation for this class was generated from the following files:

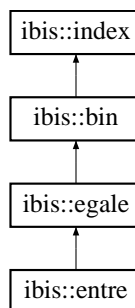
- [ibin.h](#)
- [icegale.cpp](#)

3.47 `ibis::entre` Class Reference

The multicomponent interval code on bins.

```
#include <ibin.h>
```

Inheritance diagram for `ibis::entre`:

**Public Member Functions**

- long [append](#) (const [array_t](#)< uint32_t > &ind)

Append a list of integers representing bin numbers.

- long **append** (const [ibis::entre](#) &tail)
- virtual long **append** (const char *dt, const char *df, uint32_t nnew)

Extend the index.

- **entre** (const [ibis::bin](#) &rhs, const uint32_t nbase=2)
- **entre** (const [ibis::column](#) *c, [ibis::fileManager::storage](#) *st, uint32_t offset=8)
- **entre** (const [ibis::column](#) *c=0, const char *f=0, const uint32_t nbase=2)
- virtual uint32_t **estimate** (const [ibis::qContinuousRange](#) &expr) const

Returns an upper bound on the number of hits.

- virtual void **estimate** (const [ibis::qContinuousRange](#) &expr, [ibis::bitvector](#) &lower, [ibis::bitvector](#) &upper) const

Computes an approximation of hits as a pair of lower and upper bounds.

- virtual long **evaluate** (const [ibis::qContinuousRange](#) &expr, [ibis::bitvector](#) &hits) const

To evaluate the exact hits.

- virtual double **getSum** () const

Compute the approximate sum of all the values indexed.

- virtual const char * **name** () const

Returns the name of the index, similar to the function `type`, but returns a string instead.

- virtual void **print** (std::ostream &out) const

Prints human readable information.

- virtual void **speedTest** (std::ostream &out) const

Time some logical operations and print out their speed.

- virtual INDEX_TYPE **type** () const

Returns an index type identifier.

- virtual void **write** (const char *dt) const

Save index to a file.

Protected Member Functions

- virtual double **computeSum** () const

3.47.1 Detailed Description

The multicomponent interval code on bins.

Entre is a French word for 'in between'.

3.47.2 Member Function Documentation

3.47.2.1 `void ibis::entre::estimate (const ibis::qContinuousRange & expr, ibis::bitvector & lower, ibis::bitvector & upper) const` `[virtual]`

Computes an approximation of hits as a pair of lower and upper bounds.

Parameters:

expr the query expression to be evaluated.

lower a bitvector marking a subset of the hits. All rows marked with one (1) are definitely hits.

upper a bitvector marking a superset of the hits. All hits are marked with one, but some of the rows marked one may not be hits. If the variable *upper* is empty, the variable *lower* is assumed to contain the exact answer.

Reimplemented from [ibis::egale](#).

3.47.2.2 `long ibis::entre::evaluate (const ibis::qContinuousRange & expr, ibis::bitvector & hits) const` `[virtual]`

To evaluate the exact hits.

On success, return the number of hits, otherwise a negative value is returned.

Reimplemented from [ibis::egale](#).

3.47.2.3 `double ibis::entre::getSum () const` `[virtual]`

Compute the approximate sum of all the values indexed.

If it decides that computing the sum directly from the vertical partition is more efficient, it will return NaN immediately.

Reimplemented from [ibis::egale](#).

3.47.2.4 `void ibis::entre::print (std::ostream & out) const` `[virtual]`

Prints human readable information.

Outputs information about the index as text to the specified output stream.

Reimplemented from [ibis::egale](#).

3.47.2.5 `void ibis::entre::write (const char * dt) const` `[virtual]`

Save index to a file.

Outputs the index in a compact binary format to the named file or directory. The index file contains a header that can be identified by the function `isIndex`.

Reimplemented from [ibis::egale](#).

The documentation for this class was generated from the following files:

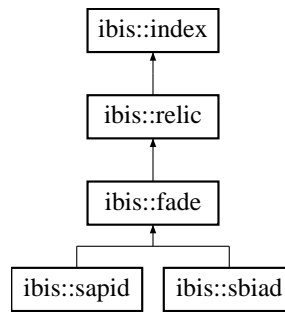
- [ibin.h](#)
- [icentre.cpp](#)

3.48 `ibis::fade` Class Reference

The multicomponent range-encoded index.

```
#include <irelic.h>
```

Inheritance diagram for `ibis::fade`:



Public Member Functions

- virtual long `append` (const char *dt, const char *df, uint32_t nnew)
Extend the index.
- virtual void `binWeights` (std::vector< uint32_t > &b) const
- virtual uint32_t `estimate` (const `ibis::qContinuousRange` &expr) const
Returns an upper bound on the number of hits.
- virtual double `estimateCost` (const `ibis::qContinuousRange` &expr) const
Estimate the code of evaluate a range condition.
- virtual long `evaluate` (const `ibis::qDiscreteRange` &expr, `ibis::bitvector` &hits) const
- virtual long `evaluate` (const `ibis::qContinuousRange` &expr, `ibis::bitvector` &hits) const
To evaluate the exact hits.
- `fade` (const `ibis::column` *c, `ibis::fileManager::storage` *st, uint32_t offset=8)
- `fade` (const `ibis::column` *c=0, const char *f=0, const uint32_t nbase=2)
- virtual double `getSum` () const
Compute the approximate sum of all the values indexed.
- virtual const char * `name` () const
Returns the name of the index, similar to the function `type`, but returns a string instead.
- virtual void `print` (std::ostream &out) const
Prints human readable information.
- virtual void `read` (`ibis::fileManager::storage` *st)
Reconstruct an index from a piece of consecutive memory.
- virtual void `read` (const char *idxfile)
Read the index contained in the file named `f`.
- virtual void `speedTest` (std::ostream &out) const
Time some logical operations and print out their speed.
- virtual INDEX_TYPE `type` () const
Returns an index type identifier.
- virtual void `write` (const char *dt) const
Save index to a file.

Protected Member Functions

- virtual void `clear` ()
Clear the existing content.
- void `write` (int fdes) const

Protected Attributes

- `array_t< uint32_t > bases`
- `array_t< uint32_t > cnts`

3.48.1 Detailed Description

The multicomponent range-encoded index.
Defined by Chan and Ioannidis (SIGMOD 98).

3.48.2 Member Function Documentation

3.48.2.1 long `ibis::fade::evaluate` (const `ibis::qContinuousRange` & *expr*, `ibis::bitvector` & *hits*) const
[virtual]

To evaluate the exact hits.

On success, return the number of hits, otherwise a negative value is returned.

Reimplemented from `ibis::relic`.

Reimplemented in `ibis::sbiad`, and `ibis::sapid`.

3.48.2.2 double `ibis::fade::getSum` () const [virtual]

Compute the approximate sum of all the values indexed.

If it decides that computing the sum directly from the vertical partition is more efficient, it will return NaN immediately.

Reimplemented from `ibis::relic`.

3.48.2.3 void `ibis::fade::print` (std::ostream & *out*) const [virtual]

Prints human readable information.

Outputs information about the index as text to the specified output stream.

Reimplemented from `ibis::relic`.

Reimplemented in `ibis::sbiad`, and `ibis::sapid`.

3.48.2.4 void `ibis::fade::write` (const char * *dt*) const [virtual]

Save index to a file.

Outputs the index in a compact binary format to the named file or directory. The index file contains a header that can be identified by the function `isIndex`.

Reimplemented from `ibis::relic`.

Reimplemented in `ibis::sbiad`, and `ibis::sapid`.

The documentation for this class was generated from the following files:

- [irelic.h](#)
- `ifade.cpp`

3.49 `ibis::fileManager` Class Reference

This `fileManager` is intended to allow different objects to share the same open file.

```
#include <fileManager.h>
```

Public Types

- enum `ACCESS_PREFERENCE` { `MMAP_LARGE_FILES`, `PREFER_READ`, `PREFER_MMAP` }
Hint passed to the function `getFile`.

Public Member Functions

- void `addCleaner` (const `cleaner` *cl)
- void `clear` ()
Close all files and remove all records of them.
- void `flushDir` (const char *name)
Close all files in the named directory, but not subdirectories.
- void `flushFile` (const char *name)
Close the file, remove the record about it from the file manager.
- void `gainReadAccess` (const char *mesg) const
Obtain a read lock on the file manager.
- int `getFile` (const char *name, `storage` **st, `ACCESS_PREFERENCE` pref=`MMAP_LARGE_FILES`)
- template<typename T> int `getFile` (const char *name, `array_t`< T > &arr, `ACCESS_PREFERENCE` pref=`MMAP_LARGE_FILES`)
- int `getFile` (const char *name, `array_t`< `rid_t` > &arr)
- int `getFile` (const char *name, `array_t`< double > &arr)
- int `getFile` (const char *name, `array_t`< float > &arr)
- int `getFile` (const char *name, `array_t`< `uint64_t` > &arr)
- int `getFile` (const char *name, `array_t`< `int64_t` > &arr)
- int `getFile` (const char *name, `array_t`< `uint32_t` > &arr)
- int `getFile` (const char *name, `array_t`< `int32_t` > &arr)
- int `getFile` (const char *name, `array_t`< unsigned char > &arr)
- int `getFile` (const char *name, `array_t`< char > &arr)
Given a file name, place the content in an `array_t`<T>.
- `storage` * `getFileSegment` (const char *name, `off_t` b, `off_t` e)
Read or memory map a portion of the named file (name).
- `time_t` `iBeat` () const
Returns the value of simple counter.
- const double & `pageCount` () const
Returns the number of pages accessed by function `read` from `stdlib.h`.

- void **printStatus** (std::ostream &out) const
Prints status information about the file manager.
- void **recordPages** (off_t start, off_t stop)
Given the starting and ending addresses, this function computes the number of pages involved.
- void **releaseAccess** (const char *mesg) const
Release a read lock on the file manager.
- void **removeCleaner** (const cleaner *cl)
- void **signalMemoryAvailable** () const
Signal to the file manager that some memory have been freed.
- int **tryGetFile** (const char *name, storage **st, ACCESS_PREFERENCE pref=MMAP_LARGE_FILES)
Returns 1 if there is not enough space to read the whole file into memory.
- template<typename T> int **tryGetFile** (const char *name, array_t< T > &arr, ACCESS_PREFERENCE pref=MMAP_LARGE_FILES)

Static Public Member Functions

- static long unsigned **bytesFree** ()
Return the number of bytes free.
- static long unsigned **bytesInUse** ()
Returns the number of bytes currently on records in the file manager.
- static void **decreaseUse** (size_t dec, const char *evt)
- static void **increaseUse** (size_t inc, const char *evt)
- static **fileManager** & **instance** ()
Returns a pointer to the one and only file manager.
- static uint32_t **pageSize** ()
Returns the page size (in bytes) used by the file system.

Protected Member Functions

- **fileManager** (const **fileManager** &rhs)
- const **fileManager** & **operator=** (const **fileManager** &rhs)
- void **recordFile** (roFile *)

Static Protected Attributes

- static unsigned long **maxBytes**
Maximum number of bytes allowed.
- static unsigned int **maxOpenFiles**
Maximum number of files opened.
- static volatile unsigned long **totalBytes**
Total bytes of all managed objects.

Friends

- class `mutexLock`
- class `roFile`
- class `storage`
- class `writeLock`

Classes

- class `cleaner`
A function object to be used to register external cleaners.
- class `mutexLock`
Used to prevent simultaneous modification of the two internal lists.
- class `readLock`
An object who uses a file under the management of the file manager should hold a `readLock`.
- class `roFile`
This class manages content of a whole (read-only) file.
- class `storage`
The storage class treats all memory as `char`.*
- class `writeLock`
A write lock for controlling access to the two interval lists.

3.49.1 Detailed Description

This `fileManager` is intended to allow different objects to share the same open file. It does not manage writing of files.

3.49.2 Member Enumeration Documentation

3.49.2.1 enum `ibis::fileManager::ACCESS_PREFERENCE`

Hint passed to the function `getFile`.

The main choice is whether to use memory map or use the read function to access the content of a file.

3.49.3 Member Function Documentation

3.49.3.1 `int ibis::fileManager::getFile (const char * name, array_t< char > & arr)`

Given a file name, place the content in an `array_t<T>`.

The return value is zero (0) if the function is successful, otherwise returns a non-zero value.

3.49.3.2 `ibis::fileManager::storage * ibis::fileManager::getFileSegment (const char * name, off_t b, off_t e)`

Read or memory map a portion of the named file (`name`).

The file is not tracked by tracking the name through a separate variable in the class `rofSegment`.

3.49.3.3 `void ibis::fileManager::recordPages (off_t start, off_t stop)` [inline]

Given the starting and ending addresses, this function computes the number of pages involved.

Used by derived classes to record page accesses.

3.49.3.4 `int ibis::fileManager::tryGetFile (const char * name, storage ** st, ACCESS_PREFERENCE pref = MMAP_LARGE_FILES)`

Returns 1 if there is not enough space to read the whole file into memory.

Other return values are same as the function `getFile`.

The documentation for this class was generated from the following files:

- [fileManager.h](#)
- [fileManager.cpp](#)

3.50 `ibis::fileManager::cleaner` Class Reference

A function object to be used to register external cleaners.

```
#include <fileManager.h>
```

Public Member Functions

- virtual void `operator() () const=0`

3.50.1 Detailed Description

A function object to be used to register external cleaners.

The documentation for this class was generated from the following file:

- [fileManager.h](#)

3.51 `ibis::fileManager::readLock` Class Reference

An object who uses a file under the management of the file manager should hold a [readLock](#).

```
#include <fileManager.h>
```

Public Member Functions

- `readLock (const char *m)`

3.51.1 Detailed Description

An object who uses a file under the management of the file manager should hold a [readLock](#).

The documentation for this class was generated from the following file:

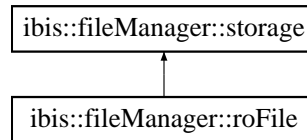
- [fileManager.h](#)

3.52 `ibis::fileManager::roFile` Class Reference

This class manages content of a whole (read-only) file.

```
#include <fileManager.h>
```

Inheritance diagram for `ibis::fileManager::roFile`:



Public Member Functions

- virtual void **beginUse** ()
- virtual void **endUse** ()
- virtual bool **isFileMap** () const
Is the storage a file map ?
- virtual size_t **printStatus** (std::ostream &out) const
- void **read** (const char *file)
- float **score** () const
The function to give score to a file.

Protected Member Functions

- virtual void **clear** ()
- void **doRead** (const char *file, off_t b, off_t e)
- void **doRead** (const char *file)
- size_t **printBody** (std::ostream &out) const

Friends

- class `ibis::fileManager`

3.52.1 Detailed Description

This class manages content of a whole (read-only) file.

It inherits the basic information stored in `fileManager::storage` and is intended to process read-only files.

3.52.2 Member Function Documentation

3.52.2.1 float `ibis::fileManager::roFile::score` () const `[inline]`

The function to give score to a file.

Files with smallest scores are the target for removal.

The documentation for this class was generated from the following files:

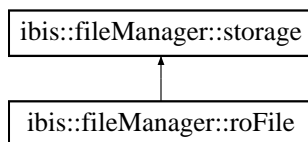
- `fileManager.h`
- `fileManager.cpp`

3.53 `ibis::fileManager::storage` Class Reference

The storage class treats all memory as `char*`.

```
#include <fileManager.h>
```

Inheritance diagram for `ibis::fileManager::storage`:



Public Member Functions

- `const char * begin () const`
- `char * begin ()`
Starting address of the storage object.
- `virtual void beginUse ()`
- `size_t bytes () const`
Return the number of bytes contained in the object.
- `void copy (const storage &rhs)`
- `bool empty () const`
Is the storage object empty?
- `const char * end () const`
- `virtual void endUse ()`
- `void enlarge (size_t nelm=0)`
Enlarge the current array by 61.8% if `nelm` is smaller than the current size, otherwise enlarge to the specified size.
- `const char * filename () const`
- `unsigned inUse () const`
- `virtual bool isFileMap () const`
Is the storage a file map ?
- `const storage & operator= (const storage &rhs)`
- `unsigned pastUse () const`
- `virtual size_t printStatus (std::ostream &out) const`
- `off_t read (const int fdes, const off_t begin, const off_t end)`
Read part of an open file. Return the number of bytes read.
- `size_t size () const`
Return the size (bytes) of the object.
- `storage (const char *begin, const char *end)`
Make another copy of the content in the range [begin, end).
- `storage (const storage &rhs)`
- `storage (const int fdes, const off_t begin, const off_t end)`
Read part of a file [start, end).

- `storage` (`size_t n`)
- void `swap` (`storage &rhs`) `throw ()`
- int `unnamed` ()

Those storage not associated with files do not have names.

- void `write` (`const char *file`) `const`
Write the content to the named file.

Protected Member Functions

- virtual void `clear` ()

Protected Attributes

- `char * m_begin`
- `char * m_end`
- unsigned `nacc`
- `char * name`
- unsigned `nref`

3.53.1 Detailed Description

The `storage` class treats all memory as `char*`.

It only uses `malloc` family of functions to manage the memory allocation and deallocation.

3.53.2 Member Function Documentation

3.53.2.1 `unsigned ibis::fileManager::storage::inUse () const` `[inline]`

3.53.2.2 `unsigned ibis::fileManager::storage::pastUse () const` `[inline]`

3.53.2.3 `int ibis::fileManager::storage::unnamed ()` `[inline]`

Those storage not associated with files do not have names.

They are not tracked by the file manager and should be immediately freed after use.

The documentation for this class was generated from the following files:

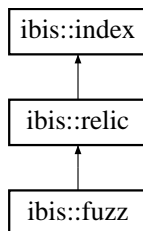
- [fileManager.h](#)
- [fileManager.cpp](#)

3.54 `ibis::fuzz` Class Reference

The two-level interval-equality code.

```
#include <irelic.h>
```

Inheritance diagram for `ibis::fuzz`:



Public Member Functions

- virtual long `append` (const char *dt, const char *df, uint32_t nnew)
Extend the index.
- virtual uint32_t `estimate` (const `ibis::qContinuousRange` &expr) const
Returns an upper bound on the number of hits.
- virtual double `estimateCost` (const `ibis::qContinuousRange` &expr) const
Estimate the code of evaluate a range condition.
- virtual long `evaluate` (const `ibis::qContinuousRange` &expr, `ibis::bitvector` &hits) const
To evaluate the exact hits.
- `fuzz` (const `ibis::column` *c, `ibis::fileManager::storage` *st, uint32_t offset=8)
The leading portion of the index file is the same as `ibis::relic`, which allows the constructor of the base class to work properly.
- `fuzz` (const `ibis::column` *c=0, const char *f=0)
- virtual const char * `name` () const
Returns the name of the index, similar to the function `type`, but returns a string instead.
- virtual void `print` (std::ostream &out) const
Prints human readable information.
- virtual void `read` (`ibis::fileManager::storage` *st)
Reconstructs an index from an array of bytes.
- virtual void `read` (const char *idxfile)
Reconstructs an index from the named file.
- virtual INDEX_TYPE `type` () const
Returns an index type identifier.
- virtual void `write` (const char *dt) const
Save index to a file.

Protected Member Functions

- virtual void `clear` ()
Clear the existing content.

3.54.1 Detailed Description

The two-level interval-equality code.

Note:

In fuzzy classification / clustering, many interval equality conditions are used. Hence the crazy name.

3.54.2 Constructor & Destructor Documentation

3.54.2.1 `ibis::fuzz::fuzz (const ibis::column * c, ibis::fileManager::storage * st, uint32_t start = 8)`

The leading portion of the index file is the same as [ibis::relic](#), which allows the constructor of the base class to work properly.

The content following the last bitvector in [ibis::relic](#) is as follows, `writeCoarse`.

`nc` (`uint32_t`) – number of coarse bins. `cbounds` (`unsigned[nc+1]`) – boundaries of the coarse bins. `coffsets` (`int32_t[nc-ceil(nc/2)+2]`) – starting positions. `cbits` (`bitvector[nc-ceil(nc/2)+1]`) – bitvectors.

3.54.3 Member Function Documentation

3.54.3.1 `long ibis::fuzz::evaluate (const ibis::qContinuousRange & expr, ibis::bitvector & hits) const` `[virtual]`

To evaluate the exact hits.

On success, return the number of hits, otherwise a negative value is returned.

Reimplemented from [ibis::relic](#).

3.54.3.2 `void ibis::fuzz::print (std::ostream & out) const` `[virtual]`

Prints human readable information.

Outputs information about the index as text to the specified output stream.

Reimplemented from [ibis::relic](#).

3.54.3.3 `void ibis::fuzz::read (ibis::fileManager::storage * st)` `[virtual]`

Reconstructs an index from an array of bytes.

Intended for internal use only!

Reimplemented from [ibis::relic](#).

3.54.3.4 `void ibis::fuzz::read (const char * idxfile)` `[virtual]`

Reconstructs an index from the named file.

The name can be the directory containing an index file. In this case, the name of the index file must be the name of the column followed by ".idx" suffix.

Reimplemented from [ibis::relic](#).

3.54.3.5 `void ibis::fuzz::write (const char * dt) const` `[virtual]`

Save index to a file.

Outputs the index in a compact binary format to the named file or directory. The index file contains a header that can be identified by the function `isIndex`.

Reimplemented from [ibis::relic](#).

The documentation for this class was generated from the following files:

- [irelic.h](#)
- [ixfuzz.cpp](#)

3.55 `ibis::horometer` Class Reference

Horometer – a primitive timing instrument.

```
#include <horometer.h>
```

Public Member Functions

- double [CPUTime](#) () const
Return the CPU time in seconds.
- double [realTime](#) () const
Return the elapsed time in seconds.
- void [resume](#) ()
- void [start](#) ()
- void [stop](#) ()

3.55.1 Detailed Description

Horometer – a primitive timing instrument.

This is intended to be a simple timer. It must be explicitly started by calling function `start`. The same function `start` may be called to restart the timer which will discard previously marked starting point. The function `stop` must be called before functions `realTime` and `CPUTime` can report the correct time. After a horometer is stopped, it may restart by calling `start`, or it may resume timing by calling `resume`.

Timing accuracy depends on the underlying implementation. On most unix systems, the time resolution is about 0.01 seconds. The timing function itself may take ~10,000 clock cycles to execute, which is about 30 microseconds on a 400 MHz machine.

3.55.2 Member Function Documentation

3.55.2.1 `void ibis::horometer::resume ()` [inline]

3.55.2.2 `void ibis::horometer::start ()` [inline]

3.55.2.3 `void ibis::horometer::stop ()` [inline]

The documentation for this class was generated from the following file:

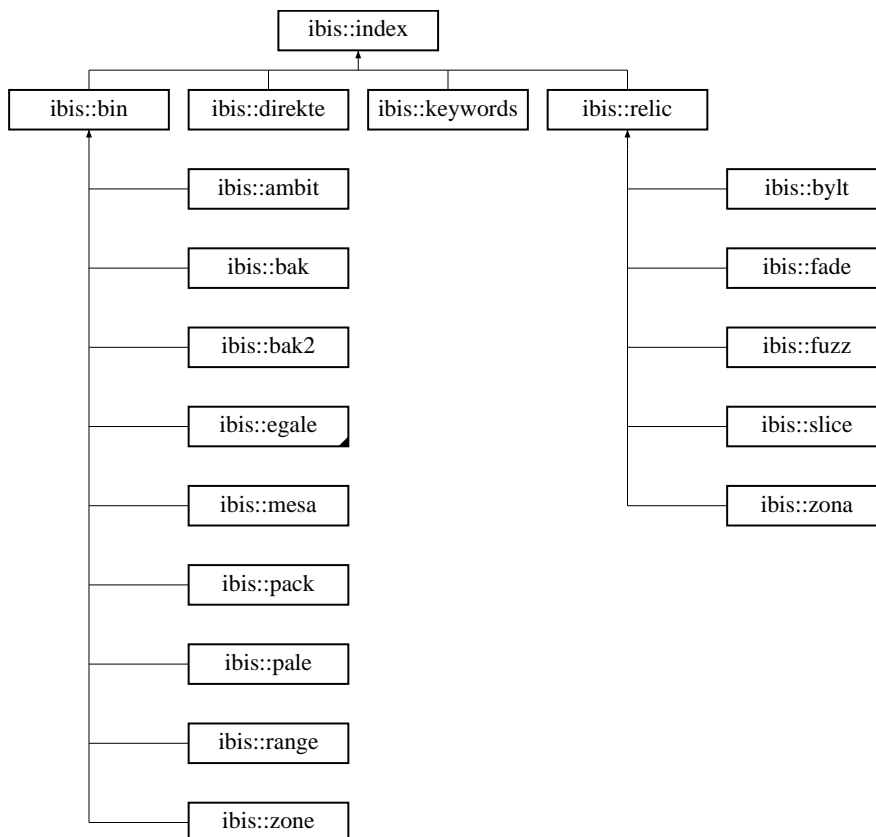
- [horometer.h](#)

3.56 `ibis::index` Class Reference

The base index class.

```
#include <index.h>
```

Inheritance diagram for `ibis::index`:



Public Types

- typedef `std::map< double, uint32_t >` **histogram**
- enum `INDEX_TYPE` {
BINNING = 0, RANGE, MESA, AMBIT,
PALE, PACK, ZONE, RELIC,
ROSTER, SLICE, FADE, SBIAD,
SAPID, EGALE, MOINS, ENTRE,
BAK, BAK2, KEYWORDS, MESH,
BAND, DIREKTE, GENERIC, BYLT,
FUZZ, ZONA }
- typedef `std::map< double, ibis::bitvector * >` **VMap**

Public Member Functions

- virtual long [append](#) (const char *dt, const char *df, uint32_t nnew)=0
Extend the index.

- virtual void **binBoundaries** (std::vector< double > &) const=0
The function binBoundaries and binWeights return bin boundaries and counts of each bin respectively.
- virtual void **binWeights** (std::vector< uint32_t > &) const=0
- virtual int **contractRange** (ibis::qContinuousRange &rng) const
- virtual int64_t **estimate** (const ibis::rangeJoin &expr, const ibis::bitvector &mask, const ibis::qRange *const range1, const ibis::qRange *const range2) const
- virtual void **estimate** (const ibis::rangeJoin &expr, const ibis::bitvector &mask, const ibis::qRange *const range1, const ibis::qRange *const range2, ibis::bitvector64 &lower, ibis::bitvector64 &upper) const
Evaluating a join condition with one (likely composite) index.
- virtual int64_t **estimate** (const ibis::index &idx2, const ibis::rangeJoin &expr, const ibis::bitvector &mask, const ibis::qRange *const range1, const ibis::qRange *const range2) const
- virtual int64_t **estimate** (const ibis::index &idx2, const ibis::rangeJoin &expr, const ibis::bitvector &mask) const
Estimate an upper bound for the number of pairs produced from marked records.
- virtual int64_t **estimate** (const ibis::index &idx2, const ibis::rangeJoin &expr) const
Estimate an upper bound for the number of pairs.
- virtual void **estimate** (const ibis::index &idx2, const ibis::rangeJoin &expr, const ibis::bitvector &mask, const ibis::qRange *const range1, const ibis::qRange *const range2, ibis::bitvector64 &lower, ibis::bitvector64 &upper) const
- virtual void **estimate** (const ibis::index &idx2, const ibis::rangeJoin &expr, const ibis::bitvector &mask, ibis::bitvector64 &lower, ibis::bitvector64 &upper) const
Estimate the pairs for the range join operator.
- virtual void **estimate** (const ibis::index &idx2, const ibis::rangeJoin &expr, ibis::bitvector64 &lower, ibis::bitvector64 &upper) const
Estimate the pairs for the range join operator.
- virtual uint32_t **estimate** (const ibis::qDiscreteRange &expr) const
- virtual void **estimate** (const ibis::qDiscreteRange &expr, ibis::bitvector &lower, ibis::bitvector &upper) const
Estimate the hits for discrete ranges, i.e., those translated from 'a IN (x, y, .
- virtual uint32_t **estimate** (const ibis::qContinuousRange &expr) const=0
Returns an upper bound on the number of hits.
- virtual void **estimate** (const ibis::qContinuousRange &expr, ibis::bitvector &lower, ibis::bitvector &upper) const =0
Computes an approximation of hits as a pair of lower and upper bounds.
- virtual double **estimateCost** (const ibis::qDiscreteRange &expr) const=0
- virtual double **estimateCost** (const ibis::qContinuousRange &expr) const=0
Estimate the code of evaluate a range condition.
- virtual long **evaluate** (const ibis::qContinuousRange &expr, ibis::bitvector &hits) const =0
To evaluate the exact hits.
- virtual int **expandRange** (ibis::qContinuousRange &rng) const
The functions expandRange and contractRange expands or contracts the boundaries of a range condition so that the new range will have exact answers using the function estimate.
- virtual const ibis::bitvector * **getBitvector** (uint32_t i) const

*Return a pointer to the *i*th bitvector used in the index (may be 0).*

- virtual long **getCumulativeDistribution** (std::vector< double > &bds, std::vector< uint32_t > &cts) const
Cumulative distribution of the data.
- virtual long **getDistribution** (std::vector< double > &bbs, std::vector< uint32_t > &cts) const
Binned distribution of the data.
- virtual double **getMax** () const=0
The maximum value recorded in the index.
- virtual double **getMin** () const=0
The minimum value recorded in the index.
- virtual uint32_t **getNRows** () const
- virtual double **getSum** () const=0
Compute the approximate sum of all the values indexed.
- virtual const char * **name** () const=0
Returns the name of the index, similar to the function `type`, but returns a string instead.
- virtual uint32_t **numBitvectors** () const
Returns the number of bit vectors used by the index.
- virtual void **print** (std::ostream &out) const=0
Prints human readable information.
- virtual void **read** (ibis::fileManager::storage *st)=0
Reconstructs an index from an array of bytes.
- virtual void **read** (const char *name)=0
Reconstructs an index from the named file.
- virtual void **speedTest** (std::ostream &out) const
Time some logical operations and print out their speed.
- virtual INDEX_TYPE **type** () const=0
Returns an index type identifier.
- virtual float **undecidable** (const ibis::qDiscreteRange &expr, ibis::bitvector &iffy) const
- virtual float **undecidable** (const ibis::qContinuousRange &expr, ibis::bitvector &iffy) const =0
Mark the position of the rows that can not be decided with this index.
- virtual void **write** (const char *name) const=0
Save index to a file.
- virtual **~index** ()
The destructor.

Static Public Member Functions

- static **ibis::index** * **create** (const **ibis::column** *c, const char *name=0, const char *spec=0)
Index factory.
- static void **divideCounts** (**array_t**< uint32_t > &bounds, const **array_t**< uint32_t > &cnt)
Determine how to split the array cnt, so that each group has roughly the same total value.
- static bool **isIndex** (const char *f, INDEX_TYPE t)
Read the header of the named file to determine if it contains an index of the specified type.
- template<typename E1, typename E2> static void **mapValues** (const **array_t**< E1 > &val1, const **array_t**< E2 > &val2, **array_t**< E1 > &bnd1, **array_t**< E2 > &bnd2, std::vector< uint32_t > &cnts)
Compute a two-dimensional histogram.
- template<typename E> static void **mapValues** (const **array_t**< E > &val, **array_t**< E > &bounds, std::vector< uint32_t > &cnts)
- template<typename E> static void **mapValues** (const **array_t**< E > &val, histogram &hist, uint32_t count=0)
- template<typename E> static void **mapValues** (const **array_t**< E > &val, VMap &bmap)

Protected Member Functions

- virtual void **activate** (uint32_t i, uint32_t j) const
Regenerate bitvectors i (inclusive) through j (exclusive) from the underlying storage.
- virtual void **activate** (uint32_t i) const
Regenerate the ith bitvector from the underlying storage.
- virtual void **activate** () const
Regenerate all bitvectors from the underlying storage.
- void **addBits** (uint32_t ib, uint32_t ie, **ibis::bitvector** &res, const **ibis::bitvector** &tot) const
Compute the sum of bit vectors [ib, ie).
- void **addBits** (uint32_t ib, uint32_t ie, **ibis::bitvector** &res) const
Add the sum of bits[ib] through bits[ie-1] to res.
- virtual void **clear** ()
Clear the existing content.
- void **computeMinMax** (const char *f, double &min, double &max) const
- void **dataFileName** (const char *f, std::string &name) const
Generate data file name from "f".
- **index** (const **ibis::column** *c=0, **ibis::fileManager::storage** *s=0)
*Protect the constructors so that **ibis::index** can not be instantiated directly.*
- void **indexFileName** (const char *f, std::string &name) const
Generate index file name from "f".
- void **mapValues** (const char *f, histogram &hist, uint32_t count=0) const
Generate a histogram.

- void `mapValues` (const char *f, VMap &bmap) const
Map the positions of each individual value.
- void `optionalUnpack` (std::vector< `ibis::bitvector` * > &bits, const char *opt)
A function to decide whether to uncompress the bitvectors.
- void `sumBits` (uint32_t ib, uint32_t ie, `ibis::bitvector` &res, uint32_t ib0, uint32_t ie0) const
Compute a new sum for bit vectors [ib, ie) by taking advantage of the old sum for bitvectors [ib0, ie0).
- void `sumBits` (uint32_t ib, uint32_t ie, `ibis::bitvector` &res) const
Compute the bitwise OR of all bitvectors (in bits) from ib to ie.

Static Protected Member Functions

- static void `addBins` (const std::vector< `ibis::bitvector` * > &bits, uint32_t ib, uint32_t ie, `ibis::bitvector` &res)
Add the bits[ib:ie-1] to res.
- static void `indexFileName` (std::string &name, const `ibis::column` *col1, const `ibis::column` *col2, const char *f=0)
Generate the index file name for the composite index formed on two columns.
- static void `setBases` (`array_t`< uint32_t > &bases, uint32_t card, uint32_t nbase=2)
Fill the array bases with the values that cover the range [0, card).
- static void `sumBins` (const std::vector< `ibis::bitvector` * > &bits, const `ibis::bitvector` &tot, uint32_t ib, uint32_t ie, `ibis::bitvector` &res)
Sum up bits[ib:ie-1] and add the result to res.
- static void `sumBins` (const std::vector< `ibis::bitvector` * > &bits, uint32_t ib, uint32_t ie, `ibis::bitvector` &res)
Sum up bits[ib:ie-1] and place the result in res.

Protected Attributes

- std::vector< `ibis::bitvector` * > `bits`
A list of bitvectors.
- const `ibis::column` * `col`
Pointer to the column this index is for.
- const char * `fname`
The name of the file containing the index.
- uint32_t `nrows`
The number of rows represented by the index.
- `array_t`< int32_t > `offsets`
Starting positions of the bitvectors.
- `ibis::fileManager::storage` * `str`
The underlying storage.

Classes

- class `barrel`

A specialization that adds function `setValue`.

3.56.1 Detailed Description

The base index class.

Class `ibis::index` contains the common definitions and virtual functions of the class hierarchy. It is assumed that an `ibis::index` is for only one column. The user is to create a new index through the function `ibis::index::create` and only use the functions defined in this class.

3.56.2 Constructor & Destructor Documentation

3.56.2.1 `ibis::index::index (const ibis::column * c = 0, ibis::fileManager::storage * s = 0)` [`inline`, `protected`]

Protect the constructors so that `ibis::index` can not be instantiated directly.

It already can not be instantiated because some functions are pure virtual, but this also reduces the size of public interface.

3.56.3 Member Function Documentation

3.56.3.1 `void ibis::index::addBins (const std::vector< ibis::bitvector * > & bts, uint32_t ib, uint32_t ie, ibis::bitvector & res)` [`static`, `protected`]

Add the `bts[ib:ie-1]` to `res`.

Since the set of bit vectors are explicitly given, there is no need to perform activation. To minimize the burden of deciding which bit vectors to activate, this function always use the `bts[ib]` through `bts[ie-1]`.

Note:

The caller need to activate the bit vectors!

This function still has to check whether a particular `bts[i]` is a null pointer before using the bit vector.

3.56.3.2 `void ibis::index::addBits (uint32_t ib, uint32_t ie, ibis::bitvector & res, const ibis::bitvector & tot)` `const` [`protected`]

Compute the sum of bit vectors [`ib`, `ie`).

If computing a complement is faster, assume all bit vectors add up to `tot`.

3.56.3.3 `void ibis::index::addBits (uint32_t ib, uint32_t ie, ibis::bitvector & res)` `const` [`protected`]

Add the sum of `bits[ib]` through `bits[ie-1]` to `res`.

Always explicitly use `bits[ib]` through `bits[ie-1]`.

3.56.3.4 `ibis::index * ibis::index::create (const ibis::column * c, const char * name = 0, const char * spec = 0)` [`static`]

Index factory.

It creates a specific concrete index object. If this function failed to read a specified index file, it will attempt to create a new index based on the current data file and index specification. The new index will be written under the old name.

This function returns nil if it fails to create an index.

Parameters:

- c* a pointer to a `ibis::column` object. This argument must be present.
- name* a name, it can be the name of the index file, the data file, or the directory containing the data file. If the name ends with `.idx` is treated as an index file and the content of the file is read. If the name does not end with `.idx`, it is assumed to be the data file name unless it is determined to be directory name. If it is a directory name, the data file is assumed to be in the directory with the file name same as the column name. Once a data file is found, the content of the data file is read to construct a new index according to the return value of function `indexSpec`. The argument *name* can be nil, in which case, the data file name is constructed by concatenate the return of `dataTable()->currentDataDir()` and the column name.

Note:

Set *name* to null to build a brand new index and discard the existing index.

Parameters:

- spec* the index specification. This string contains the parameters for how to create an index. The most general form is

```
<binning .../> <encoding .../> <compression .../>.
```

Here is one example (it is the default for some integer columns)

```
<binning none /> <encoding equality />
```

FastBit always compresses every bitmap it ever generates. The compression option is to instruct it to uncompress some bitmaps or not compress indices at all. The compress option is usually not used.

If the argument *spec* is not specified, this function checks the specification in the following order.

1. use the index specification for the column being indexed;
2. use the index specification for the table containing the column being indexed;
3. use the most specific index specification relates to the column be indexed in the global resources (`gParameters`).

It stops looking as soon as it finds the first non-empty string. To override any general index specification, one must provide a complete index specification string.

3.56.3.5 `void ibis::index::divideCounts (array_t< uint32_t > & bdry, const array_t< uint32_t > & cnt)` [static]

Determine how to split the array *cnt*, so that each group has roughly the same total value.

The first group is `[0, bdry[0])`, the second group is `[bdry[0], bdry[1])`, and so on. This function uses the size of array *bdry* to determine the number of groups to use.

3.56.3.6 `void ibis::index::estimate (const ibis::index & idx2, const ibis::rangeJoin & expr, const ibis::bitvector & mask, ibis::bitvector64 & lower, ibis::bitvector64 & upper) const` [virtual]

Estimate the pairs for the range join operator.

Only records that are masked are evaluated.

3.56.3.7 `void ibis::index::estimate (const ibis::qDiscreteRange & expr, ibis::bitvector & lower, ibis::bitvector & upper) const` [virtual]

Estimate the hits for discrete ranges, i.e., those translated from 'a IN (x, y, . . .)'.
 Reimplemented in `ibis::direkte`, and `ibis::relic`.

3.56.3.8 `virtual void ibis::index::estimate (const ibis::qContinuousRange & expr, ibis::bitvector & lower, ibis::bitvector & upper) const` [pure virtual]

Computes an approximation of hits as a pair of lower and upper bounds.

Parameters:

expr the query expression to be evaluated.

lower a bitvector marking a subset of the hits. All rows marked with one (1) are definitely hits.

upper a bitvector marking a superset of the hits. All hits are marked with one, but some of the rows marked one may not be hits. If the variable upper is empty, the variable lower is assumed to contain the exact answer.

Implemented in `ibis::bin`, `ibis::range`, `ibis::mesa`, `ibis::ambit`, `ibis::pale`, `ibis::pack`, `ibis::zone`, `ibis::egale`, `ibis::moins`, `ibis::entre`, `ibis::direkte`, `ibis::keywords`, `ibis::relic`, and `ibis::slice`.

3.56.3.9 `virtual long ibis::index::evaluate (const ibis::qContinuousRange & expr, ibis::bitvector & hits) const` [pure virtual]

To evaluate the exact hits.

On success, return the number of hits, otherwise a negative value is returned.

Implemented in `ibis::bin`, `ibis::range`, `ibis::mesa`, `ibis::ambit`, `ibis::pale`, `ibis::pack`, `ibis::zone`, `ibis::egale`, `ibis::moins`, `ibis::entre`, `ibis::direkte`, `ibis::keywords`, `ibis::relic`, `ibis::slice`, `ibis::fade`, `ibis::sbiad`, `ibis::sapid`, `ibis::fuzz`, `ibis::bylt`, and `ibis::zona`.

3.56.3.10 `virtual int ibis::index::expandRange (ibis::qContinuousRange & rng) const` [inline, virtual]

The functions `expandRange` and `contractRange` expands or contracts the boundaries of a range condition so that the new range will have exact answers using the function `estimate`.

The default implementation provided does nothing since this is only meaningful for indices based on bins.

Reimplemented in `ibis::bin`, `ibis::range`, `ibis::bak`, and `ibis::bak2`.

3.56.3.11 `virtual double ibis::index::getSum () const` [pure virtual]

Compute the approximate sum of all the values indexed.

If it decides that computing the sum directly from the vertical partition is more efficient, it will return NaN immediately.

Implemented in `ibis::bin`, `ibis::range`, `ibis::mesa`, `ibis::ambit`, `ibis::pack`, `ibis::egale`, `ibis::moins`, `ibis::entre`, `ibis::direkte`, `ibis::keywords`, `ibis::relic`, `ibis::slice`, and `ibis::fade`.

3.56.3.12 `void ibis::index::indexFileName (std::string & iname, const ibis::column * coll, const ibis::column * col2, const char * dir = 0)` [static, protected]

Generate the index file name for the composite index fromed on two columns.

May use argument "dir" if it is not null.

3.56.3.13 `bool ibis::index::isIndex (const char *f, INDEX_TYPE t) [static]`

Read the header of the named file to determine if it contains an index of the specified type.

Returns true if the correct header is found, else return false.

3.56.3.14 `void ibis::index::mapValues (const char * f, histogram & hist, uint32_t count = 0) const [protected]`

Generate a histogram.

A value of any supported type is supposed to be able to fit in a double with no rounding, no approximation and no overflow.

3.56.3.15 `void ibis::index::mapValues (const char *f, VMap & bmap) const [protected]`

Map the positions of each individual value.

Given a file containing the values of a column, this function maps the position of each individual values and stores the result in a set of bitmaps.

Note:

IMPOTANT ASSUMPTION. A value of any supported type is supposed to be able to fit in a double with no rounding, no approximation and no overflow.

3.56.3.16 `template<typename E1, typename E2> void ibis::index::mapValues (const array_t< E1 > & val1, const array_t< E2 > & val2, array_t< E1 > & bnd1, array_t< E2 > & bnd2, std::vector< uint32_t > & cnts) [static]`

Compute a two-dimensional histogram.

Given two arrays of the same size, count the number of appearance of each combinations defined by `bnd1` and `bnd2`. If the arrays `bnd1` or `bnd2` contain values in ascending order, their values are directly used in this function. The empty one will be replaced by a linear division of actual range into 256 bins. The array `counts` stores the 2-D bins in raster-scan order with the second variable, `val2`, as the faster varying variables. More specifically, the bins for variable 1 are defined as:

```
/// (... , bnd1[0]) [bnd1[1], bin1[2]) [bnd1[2], bin1[3) ... [bnd1.back(), ...)
///
```

Note that '[' denote the left boundary is inclusive and ')' denote the right boundary is exclusive. Similarly, the bins for variable 2 are

```
/// (... , bnd2[0]) [bnd2[1], bin2[2]) [bnd2[2], bin2[3) ... [bnd2.back(), ...)
///
```

The `counts` are for the following bins

```
/// (... , bin1[0]) (... bin2[0])
/// (... , bin1[0]) [bin2[0], bin2[1])
/// (... , bin1[0]) [bin2[1], bin2[2])
/// ...
/// (... , bin1[0]) [bin2.back(), ...)
/// [bin1[0], bin1[1]) (... , bin2[0])
/// [bin1[0], bin1[1]) [bin2[0], bin2[1])
/// [bin1[0], bin1[1]) [bin2[1], bin2[2])
/// ...
/// [bin1[0], bin1[1]) [bin2.back(), ...)
/// ...
///
```

3.56.3.17 `virtual void ibis::index::print (std::ostream & out) const` [pure virtual]

Prints human readable information.

Outputs information about the index as text to the specified output stream.

Implemented in `ibis::bin`, `ibis::range`, `ibis::mesa`, `ibis::ambit`, `ibis::pale`, `ibis::pack`, `ibis::zone`, `ibis::egale`, `ibis::moins`, `ibis::entre`, `ibis::bak`, `ibis::bak2`, `ibis::direkte`, `ibis::keywords`, `ibis::relic`, `ibis::slice`, `ibis::fade`, `ibis::sbiad`, `ibis::sapid`, `ibis::fuzz`, `ibis::bylt`, and `ibis::zona`.

3.56.3.18 `virtual void ibis::index::read (ibis::fileManager::storage * st)` [pure virtual]

Reconstructs an index from an array of bytes.

Intended for internal use only!

Implemented in `ibis::bin`, `ibis::range`, `ibis::ambit`, `ibis::pale`, `ibis::pack`, `ibis::zone`, `ibis::egale`, `ibis::direkte`, `ibis::keywords`, `ibis::relic`, `ibis::slice`, `ibis::fade`, `ibis::fuzz`, `ibis::bylt`, and `ibis::zona`.

3.56.3.19 `virtual void ibis::index::read (const char * name)` [pure virtual]

Reconstructs an index from the named file.

The name can be the directory containing an index file. In this case, the name of the index file must be the name of the column followed by ".idx" suffix.

Implemented in `ibis::bin`, `ibis::range`, `ibis::ambit`, `ibis::pale`, `ibis::pack`, `ibis::zone`, `ibis::egale`, `ibis::bak`, `ibis::bak2`, `ibis::direkte`, `ibis::keywords`, `ibis::relic`, `ibis::slice`, `ibis::fade`, `ibis::fuzz`, `ibis::bylt`, and `ibis::zona`.

3.56.3.20 `void ibis::index::setBases (array_t< uint32_t > & bases, uint32_t card, uint32_t nbase = 2)` [static, protected]

Fill the array bases with the values that cover the range [0, card).

Assumes at least two components. For one component case use indices defined explicit for one component cases.

3.56.3.21 `void ibis::index::sumBins (const std::vector< ibis::bitvector * > & bts, const ibis::bitvector & tot, uint32_t ib, uint32_t ie, ibis::bitvector & res)` [static, protected]

Sum up `bts[ib:ie-1]` and add the result to `res`.

It is assumed that all `bts` add up to `tot`. In the other version of `sumBins` without this argument `tot`, it was assumed that all bitmaps add up to a bit vector of all ones. The decision of whether to use `bts[ib:ie-1]` directly or use the subtractive version (using `bts[0:ib-1]` and `bts[ie:nobs-1]`) are based on the number of bit vectors.

3.56.3.22 `void ibis::index::sumBins (const std::vector< ibis::bitvector * > & bts, uint32_t ib, uint32_t ie, ibis::bitvector & res)` [static, protected]

Sum up `bts[ib:ie-1]` and place the result in `res`.

Note:

This function may either use `bts[ib:ie-1]` or `bts[0:ib-1]` and `bts[ie:nobs-1]` depending which set has more bit vectors! This requires the caller to activate the appropriate set.

This function always uses the operator `|=`. Tests show that using the function `setBit` is always slower.

3.56.3.23 `void ibis::index::sumBits (uint32_t ib, uint32_t ie, ibis::bitvector & res, uint32_t ib0, uint32_t ie0) const` [protected]

Compute a new sum for bit vectors [ib, ie) by taking advantage of the old sum for bitvectors [ib0, ie0).

- On input, $res = \sum_{i=ib0}^{ie0} bits[i]$.
- On exit, $res = \sum_{i=ib}^{ie} bits[i]$.

3.56.3.24 `void ibis::index::sumBits (uint32_t ib, uint32_t ie, ibis::bitvector & res) const` [protected]

Compute the bitwise OR of all bitvectors (in bits) from `ib` to `ie`.

As usual, `bits[ib]` is included but `bits[ie]` is excluded.

3.56.3.25 `virtual float ibis::index::undecidable (const ibis::qContinuousRange & expr, ibis::bitvector & iffy) const` [pure virtual]

Mark the position of the rows that can not be decided with this index.

Parameters:

- expr* the range conditions to be evaluated.
- iffy* the bitvector marking the positions of rows that can not be decided using the index. Return value is the expected fraction of undecided rows that might satisfy the range conditions.

Implemented in `ibis::bin`, `ibis::range`, `ibis::mesa`, `ibis::ambit`, `ibis::pale`, `ibis::pack`, `ibis::zone`, `ibis::egale`, `ibis::direkte`, `ibis::keywords`, and `ibis::relic`.

3.56.3.26 `virtual void ibis::index::write (const char * name) const` [pure virtual]

Save index to a file.

Outputs the index in a compact binary format to the named file or directory. The index file contains a header that can be identified by the function `isIndex`.

Implemented in `ibis::bin`, `ibis::range`, `ibis::mesa`, `ibis::ambit`, `ibis::pale`, `ibis::pack`, `ibis::zone`, `ibis::egale`, `ibis::moins`, `ibis::entre`, `ibis::bak`, `ibis::bak2`, `ibis::direkte`, `ibis::keywords`, `ibis::relic`, `ibis::slice`, `ibis::fade`, `ibis::sbiad`, `ibis::sapid`, `ibis::fuzz`, `ibis::bylt`, and `ibis::zona`.

3.56.4 Member Data Documentation

3.56.4.1 `ibis::fileManager::storage* ibis::index::str` [mutable, protected]

The underlying storage.

It may be nil if bitvectors are not from a storage object managed by the file manager.

The documentation for this class was generated from the following files:

- `index.h`
- `index.cpp`

3.57 `ibis::index::barrel` Class Reference

A specialization that adds function `setValue`.

```
#include <index.h>
```

Public Member Functions

- `barrel` (const `ibis::compRange::term *`t)
- void `setValue` (uint32_t i, double v)

3.57.1 Detailed Description

A specialization that adds function `setValue`.

This function allows the client to directly set the value for an individual variable.

The documentation for this class was generated from the following file:

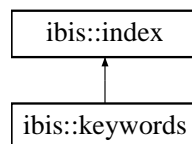
- [index.h](#)

3.58 `ibis::keywords` Class Reference

Class `ibis::keywords` defines a boolean term-document matrix.

```
#include <ikeywords.h>
```

Inheritance diagram for `ibis::keywords`:



Public Member Functions

- virtual long [append](#) (const char *dt, const char *df, uint32_t nnew)
Extend the index.
- virtual void [binBoundaries](#) (std::vector< double > &b) const
The function `binBoundaries` and `binWeights` return bin boundaries and counts of each bin respectively.
- virtual void [binWeights](#) (std::vector< uint32_t > &b) const
- virtual uint32_t [estimate](#) (const [ibis::qContinuousRange](#) &expr) const
Returns an upper bound on the number of hits.
- virtual void [estimate](#) (const [ibis::qContinuousRange](#) &expr, [ibis::bitvector](#) &lower, [ibis::bitvector](#) &upper) const
Computes an approximation of hits as a pair of lower and upper bounds.
- virtual double [estimateCost](#) (const [ibis::qDiscreteRange](#) &expr) const
- virtual double [estimateCost](#) (const [ibis::qContinuousRange](#) &expr) const
Estimate the code of evaluate a range condition.
- virtual long [evaluate](#) (const [ibis::qContinuousRange](#) &expr, [ibis::bitvector](#) &hits) const
To evaluate the exact hits.
- virtual double [getMax](#) () const
The maximum value recorded in the index.
- virtual double [getMin](#) () const
The minimum value recorded in the index.
- virtual double [getSum](#) () const
Compute the approximate sum of all the values indexed.

- **keywords** (const [ibis::column](#) *c, [ibis::fileManager::storage](#) *st, uint32_t offset=8)
- **keywords** (const [ibis::column](#) *c, const [ibis::column](#) *idcol=0, const char *f=0)
Constructor.
- virtual const char * **name** () const
Returns the name of the index, similar to the function `type`, but returns a string instead.
- virtual void **print** (std::ostream &out) const
Prints human readable information.
- virtual void **read** ([ibis::fileManager::storage](#) *st)
Reconstructs an index from an array of bytes.
- virtual void **read** (const char *idxfile)
Reconstructs an index from the named file.
- long **search** (const char *kw) const
Estimate the number of matches.
- long **search** (const char *kw, [ibis::bitvector](#) &hits) const
Match a particular keyword.
- virtual INDEX_TYPE **type** () const
Returns an index type identifier.
- virtual float **undecidable** (const [ibis::qContinuousRange](#) &expr, [ibis::bitvector](#) &iffy) const
This class and its derived classes should produce exact answers, therefore no undecidable rows.
- virtual void **write** (const char *dt) const
Write the boolean term-document matrix as two files, `xxx.terms` for the terms and `xxx.idx` for the bitmaps that marks the positions.

Protected Member Functions

- void **clear** ()
Clear the current content.
- char **readKeyword** (const char *&buf, std::string &key) const
Extract the keyword from a line of input term-document file.
- int **readLine** (std::istream &in, std::string &key, std::vector< uint32_t > &idlist, char *buf, uint32_t nbuf) const
Read and process one line from the term-document file.
- int **readTermDocFile** (const [ibis::column](#) *idcol, const char *f)
Reads a term-document list from an external file.
- uint32_t **readUInt** (const char *&buf) const
Extract the next integer in an inputline.
- void **setBits** (std::vector< uint32_t > &pos, [ibis::bitvector](#) &bvec) const

3.58.1 Detailed Description

Class `ibis::keywords` defines a boolean term-document matrix.

The terms are stored in an `ibis::dictionary` and the bits are stored in a series of bitvectors.

The current implementation requires a `.tdlist` file alongside the binary string values. In addition, it uses an entry in the `ibis.rc` file of the following form to specify the ids used in the `.tdlist` file.

```
<table-name>.<column-name>.docIDName=<id-column-name>
```

For example,

```
enrondata.subject.docIDName=mid enrondata.body.docIDName=mid
```

If an ID column is not specified, the integer IDs in the `.tdlist` file is assumed to the row number.

3.58.2 Constructor & Destructor Documentation

3.58.2.1 `ibis::keywords::keywords (const ibis::column * c, const ibis::column * idcol = 0, const char * f = 0)`

Constructor.

It first tries to read the terms (`.terms`) and the `tdmat` (`.idx`) files if they both exist. If that fails, it tries to read the `tdlist` (`.tdlist`) file generated by another program and

3.58.3 Member Function Documentation

3.58.3.1 `void ibis::keywords::estimate (const ibis::qContinuousRange & expr, ibis::bitvector & lower, ibis::bitvector & upper) const` [virtual]

Computes an approximation of hits as a pair of lower and upper bounds.

Parameters:

expr the query expression to be evaluated.

lower a bitvector marking a subset of the hits. All rows marked with one (1) are definitely hits.

upper a bitvector marking a superset of the hits. All hits are marked with one, but some of the rows marked one may not be hits. If the variable `upper` is empty, the variable `lower` is assumed to contain the exact answer.

Implements [ibis::index](#).

3.58.3.2 `long ibis::keywords::evaluate (const ibis::qContinuousRange & expr, ibis::bitvector & hits) const` [virtual]

To evaluate the exact hits.

On success, return the number of hits, otherwise a negative value is returned.

Implements [ibis::index](#).

3.58.3.3 `virtual double ibis::keywords::getSum () const` [inline, virtual]

Compute the approximate sum of all the values indexed.

If it decides that computing the sum directly from the vertical partition is more efficient, it will return NaN immediately.

Implements [ibis::index](#).

3.58.3.4 `void ibis::keywords::print (std::ostream & out) const` [virtual]

Prints human readable information.

Outputs information about the index as text to the specified output stream.

Implements [ibis::index](#).

3.58.3.5 `void ibis::keywords::read (ibis::fileManager::storage * st)` [virtual]

Reconstructs an index from an array of bytes.

Intended for internal use only!

Implements [ibis::index](#).

3.58.3.6 `void ibis::keywords::read (const char * idxfile)` [virtual]

Reconstructs an index from the named file.

The name can be the directory containing an index file. In this case, the name of the index file must be the name of the column followed by ".idx" suffix.

Implements [ibis::index](#).

3.58.3.7 `char ibis::keywords::readKeyword (const char *& buf, std::string & keyword) const` [inline, protected]

Extract the keyword from a line of input term-document file.

Returns the first non-space character after the keyword.

3.58.3.8 `int ibis::keywords::readTermDocFile (const ibis::column * idcol, const char * f)` [protected]

Reads a term-document list from a external file.

Returns the number of terms found if successful, otherwise returns a negative number to indicate error.

The documentation for this class was generated from the following files:

- [ikeywords.h](#)
- [ikeywords.cpp](#)

3.59 `ibis::lessi` Struct Reference

A case-insensitive version of less for comparing names of tables, columns, and resources.

```
#include <const.h>
```

Public Member Functions

- `bool operator() (const char *x, const char *y) const`

3.59.1 Detailed Description

A case-insensitive version of less for comparing names of tables, columns, and resources.

The documentation for this struct was generated from the following file:

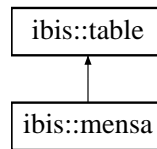
- [const.h](#)

3.60 `ibis::mena` Class Reference

Class `ibis::mena` contains multiple (horizontal) data partitions (`ibis::part`) to form a logical data table.

```
#include <mena.h>
```

Inheritance diagram for `ibis::mena`:



Public Member Functions

- virtual int `buildIndex` (const char *, const char *)
Create the index for named column.
- virtual int `buildIndexes` (const char *)
Create indexes for every column of the table.
- virtual `stringList` `columnNames` () const
Return column names.
- virtual `typeList` `columnTypes` () const
Return data types.
- virtual `ibis::table::cursor *` `createCursor` () const
Create a `CURSOR` object to perform row-wise data access.
- virtual void `describe` (std::ostream &) const
Print a description of the table to the specified output stream.
- virtual int `dump` (std::ostream &, const char *) const
Dump the values in ASCII form to the specified output stream.
- virtual void `estimate` (const char *cond, uint64_t &nmin, uint64_t &nmax) const
Estimate the number of rows satisfying the selection conditions.
- virtual int64_t `getColumnAsBytes` (const char *, char *) const
Retrieve all values of the named column.
- virtual int64_t `getColumnAsDoubles` (const char *, double *) const
Note:
Integers four-byte or less in length can be converted to double safely.
- virtual int64_t `getColumnAsFloats` (const char *, float *) const
Note:
Integers two-byte or less in length can be converted safely to floats.
- virtual int64_t `getColumnAsInts` (const char *, int32_t *) const
Retrieve all values of the named column.

- virtual `int64_t getColumnAsLongs` (`const char *`, `int64_t *`) `const`
Note:
All integers 4-byte or shorter in length can be safely converted into `int64_t`.
- virtual `int64_t getColumnAsShorts` (`const char *`, `int16_t *`) `const`
Retrieve all values of the named column.
- virtual `int64_t getColumnAsStrings` (`const char *`, `std::vector< std::string > &`) `const`
Note:
Any data type can be converted to strings, however, the conversion may take a significant amount of time.
- virtual `int64_t getColumnAsUBytes` (`const char *`, `unsigned char *`) `const`
Retrieve all values of the named column.
- virtual `int64_t getColumnAsUInts` (`const char *`, `uint32_t *`) `const`
Retrieve all values of the named column.
- virtual `int64_t getColumnAsULongs` (`const char *`, `uint64_t *`) `const`
Note:
All integers can be converted to `uint64_t`, however, negative integers will be treated as unsigned integers.
- virtual `int64_t getColumnAsUShorts` (`const char *`, `uint16_t *`) `const`
Retrieve all values of the named column.
- virtual `long getHistogram` (`const char *`, `const char *`, `double`, `double`, `double`, `std::vector< size_t > &`) `const`
Compute the histogram of the named column.
- virtual `long getHistogram2D` (`const char *`, `const char *`, `double`, `double`, `double`, `const char *`, `double`, `double`, `double`, `std::vector< size_t > &`) `const`
Compute a two-dimension histogram on columns `cname1` and `name2`.
- virtual `long getHistogram3D` (`const char *`, `const char *`, `double`, `double`, `double`, `const char *`, `double`, `double`, `double`, `double`, `double`, `std::vector< size_t > &`) `const`
Compute a three-dimensional histogram on the named columns.
- virtual `table * groupby` (`const stringList &`) `const`
Directly performing group-by on the base data (without selection) is not currently supported.
- virtual `void indexSpec` (`const char *`, `const char *`)
Replace the current indexing option.
- virtual `const char * indexSpec` (`const char *`) `const`
Retrieve the current indexing option.
- `mensa` (`const char *dir1`, `const char *dir2`)
- `mensa` (`const char *dir`)
- virtual `size_t nColumns` () `const`
- virtual `uint64_t nRows` () `const`
- virtual `void orderby` (`const stringList &`)
Reordering the rows using the specified columns.
- virtual `void reverseRows` ()

Reversing the ordering of the rows on disk requires too much work but has no obvious benefit.

- virtual [table](#) * [select](#) (const char *sel, const char *cond) const

Given a set of column names and a set of selection conditions, compute another table that represents the selected values.

Protected Member Functions

- template<typename T> void [addIncoreData](#) (void *&to, const [array_t](#)< T > &from, size_t nold, const T special) const

Append new data (in from) to a larger array (pointed to by to).

- void [addStrings](#) (void *&, const std::vector< std::string > &, size_t) const
- void [clear](#) ()

Clear the existing content.

- int64_t [computeHits](#) (const char *cond) const

Compute the number of hits.

Protected Attributes

- [ibis::table::namesTypes](#) [naty](#)

A combined list of columns names.

- uint64_t [nrows](#)
- [ibis::partList](#) [parts](#)

List of data partitions.

Friends

- class [cursor](#)

Classes

- class [cursor](#)

3.60.1 Detailed Description

Class [ibis::mensa](#) contains multiple (horizontal) data partitions ([ibis::part](#)) to form a logical data table.

The base data contained in this table is logically immutable as reordering rows (through function [reorder](#)) does not change the overall content of the table. The functions [reverseRows](#) and [groupby](#) are not implemented.

Note:

Mensa is a Latin word for "table."

3.60.2 Member Function Documentation

3.60.2.1 `int ibis::mena::buildIndex (const char *, const char *)` [virtual]

Create the index for named column.

The existing index will be replaced. If an indexing option is not specified, it will use the internally recorded option for the named column or the table containing the column.

Note:

Unless any there is a specific instruction to not index a column, the querying functions will automatically build indices as necessary. However, as building an index is relatively expensive process, building an index on a column is on average about four or five times as expensive as reading the column from disk, this function is provided so that it is possible to build indexes beforehand.

Implements [ibis::table](#).

3.60.2.2 `int ibis::mena::buildIndexes (const char *)` [virtual]

Create indexes for every column of the table.

Existing indexes will be replaced. If an indexing option is not specified, the internally recorded options will be used. [buildIndex](#)

Implements [ibis::table](#).

3.60.2.3 `int ibis::mena::dump (std::ostream &, const char *) const` [virtual]

Dump the values in ASCII form to the specified output stream.

The default delimiter is coma (","), which produces Comma-Separated-Values (CSV).

Implements [ibis::table](#).

3.60.2.4 `void ibis::mena::estimate (const char * cond, uint64_t & nmin, uint64_t & nmax) const` [virtual]

Estimate the number of rows satisfying the selection conditions.

The number of rows is between [nmin, nmax].

Implements [ibis::table](#).

3.60.2.5 `int64_t ibis::mena::getColumnAsBytes (const char *, char *) const` [virtual]

Retrieve all values of the named column.

The member functions of this class only support access to whole column at a time. Use `table::cursor` class for row-wise accesses. For fixed-width data types, the raw pointers are used to point to the values to be returned. In these cases, the caller is responsible for allocating enough storage for the values to be returned.

Implements [ibis::table](#).

3.60.2.6 `int64_t ibis::mena::getColumnAsDoubles (const char * cn, double * vals) const` [virtual]

Note:

Integers four-byte or less in length can be converted to double safely.

Float values may also be converted into doubles.

Implements [ibis::table](#).

3.60.2.7 `int64_t ibis::mensa::getColumnAsInts (const char *, int32_t *) const` [virtual]

Retrieve all values of the named column.

The member functions of this class only support access to whole column at a time. Use `table::cursor` class for row-wise accesses. For fixed-width data types, the raw pointers are used to point to the values to be returned. In these cases, the caller is responsible for allocating enough storage for the values to be returned.

Implements [ibis::table](#).

3.60.2.8 `int64_t ibis::mensa::getColumnAsLongs (const char * cn, int64_t * vals) const` [virtual]**Note:**

All integers 4-byte or shorter in length can be safely converted into `int64_t`.

Values in `uint64_t` are treated as signed integers, which may create incorrect values.

Implements [ibis::table](#).

3.60.2.9 `int64_t ibis::mensa::getColumnAsShorts (const char *, int16_t *) const` [virtual]

Retrieve all values of the named column.

The member functions of this class only support access to whole column at a time. Use `table::cursor` class for row-wise accesses. For fixed-width data types, the raw pointers are used to point to the values to be returned. In these cases, the caller is responsible for allocating enough storage for the values to be returned.

Implements [ibis::table](#).

3.60.2.10 `int64_t ibis::mensa::getColumnAsUBytes (const char *, unsigned char *) const` [virtual]

Retrieve all values of the named column.

The member functions of this class only support access to whole column at a time. Use `table::cursor` class for row-wise accesses. For fixed-width data types, the raw pointers are used to point to the values to be returned. In these cases, the caller is responsible for allocating enough storage for the values to be returned.

Implements [ibis::table](#).

3.60.2.11 `int64_t ibis::mensa::getColumnAsUInts (const char *, uint32_t *) const` [virtual]

Retrieve all values of the named column.

The member functions of this class only support access to whole column at a time. Use `table::cursor` class for row-wise accesses. For fixed-width data types, the raw pointers are used to point to the values to be returned. In these cases, the caller is responsible for allocating enough storage for the values to be returned.

Implements [ibis::table](#).

3.60.2.12 `int64_t ibis::mensa::getColumnAsUShorts (const char *, uint16_t *) const` [virtual]

Retrieve all values of the named column.

The member functions of this class only support access to whole column at a time. Use `table::cursor` class for row-wise accesses. For fixed-width data types, the raw pointers are used to point to the values to be returned. In these cases, the caller is responsible for allocating enough storage for the values to be returned.

Implements [ibis::table](#).

3.60.2.13 `long ibis::mensa::getHistogram (const char *, const char *, double, double, double, std::vector< size_t > &) const` [virtual]

Compute the histogram of the named column.

This version uses the user specified bins: $[begin, begin+stride)$ $[begin+stride, begin+2*stride)$, A record is placed in bin $(x - begin) / stride$, where the first bin is bin 0. This gives a total of

$(end - begin) / stride$ bins.

Note:

Records (rows) outside of the range $[begin, end]$ are not counted.

Non-positive `stride` is considered as an error.

If `end` is less than `begin`, an empty array `counts` is returned along with return value 0.

Implements [ibis::table](#).

3.60.2.14 `long ibis::mensa::getHistogram2D (const char *, const char *, double, double, double, const char *, double, double, double, std::vector< size_t > &) const` [virtual]

Compute a two-dimension histogram on columns `cname1` and `name2`.

The bins along each dimension are defined the same way as in function [getHistogram](#). The array `counts` stores the two-dimensional bins with the first dimension as the slow varying dimension following C convention for ordering multi-dimensional arrays.

Implements [ibis::table](#).

3.60.2.15 `long ibis::mensa::getHistogram3D (const char *, const char *, double, double, double, const char *, double, double, double, const char *, double, double, double, std::vector< size_t > &) const` [virtual]

Compute a three-dimensional histogram on the named columns.

The triplets $\langle begin, end, stride \rangle$ are used the same ways in [getHistogram](#) and [getHistogram2D](#). The three dimensional bins are linearized in `counts` with the first being the slowest varying dimension and the third being the fastest varying dimension following the C convention for ordering multi-dimensional arrays.

Implements [ibis::table](#).

3.60.2.16 `void ibis::mensa::indexSpec (const char *, const char *)` [virtual]

Replace the current indexing option.

If no column name is specified, it resets the indexing option for the table.

Implements [ibis::table](#).

3.60.2.17 `const char * ibis::mensa::indexSpec (const char *) const` [virtual]

Retrieve the current indexing option.

If no column name is specified, it retrieve the indexing option for the table.

Implements [ibis::table](#).

3.60.2.18 `void ibis::mensa::orderBy (const stringList & names)` [virtual]

Reordering the rows using the specified columns.

Each data partition is reordered separately.

Implements [ibis::table](#).

The documentation for this class was generated from the following files:

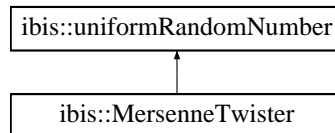
- [mensa.h](#)
- `mensa.cpp`

3.61 `ibis::MersenneTwister` Class Reference

Mersenne Twister generates uniform random numbers efficiently.

```
#include <twister.h>
```

Inheritance diagram for `ibis::MersenneTwister`:



Public Member Functions

- `MersenneTwister` (unsigned seed)
- unsigned `next` ()
- unsigned `next` (unsigned r)
- double `nextDouble` ()
- float `nextFloat` ()
- double `nextGaussian` ()
- int `nextInt` ()
- long `nextLong` ()
- double `nextPoisson` ()
- double `nextZipf` (double a)
- double `nextZipf` ()
- virtual double `operator`() ()
- void `setSeed` (unsigned seed)

3.61.1 Detailed Description

Mersenne Twister generates uniform random numbers efficiently.

The documentation for this class was generated from the following file:

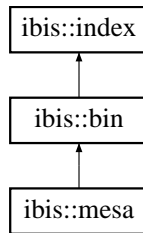
- [twister.h](#)

3.62 `ibis::mesa` Class Reference

This class implements the two-side range encoding from Chan and Ioannidis.

```
#include <ibin.h>
```

Inheritance diagram for `ibis::mesa`:



Public Member Functions

- long **append** (const [ibis::mesa](#) &tail)
- virtual long **append** (const char *dt, const char *df, uint32_t nnew)
Extend the index.
- virtual void **binBoundaries** (std::vector< double > &) const
The function binBoundaries and binWeights return bin boundaries and counts of each bin respectively.
- virtual void **binWeights** (std::vector< uint32_t > &) const
- virtual uint32_t **estimate** (const [ibis::qContinuousRange](#) &expr) const
Returns an upper bound on the number of hits.
- virtual void **estimate** (const [ibis::qContinuousRange](#) &expr, [ibis::bitvector](#) &lower, [ibis::bitvector](#) &upper) const
Computes an approximation of hits as a pair of lower and upper bounds.
- virtual long **evaluate** (const [ibis::qContinuousRange](#) &expr, [ibis::bitvector](#) &hits) const
To evaluate the exact hits.
- virtual double **getSum** () const
Compute the approximate sum of all the values indexed.
- **mesa** (const [ibis::bin](#) &rhs)
Construct an interval encoded index from an equality encoded index.
- **mesa** (const [ibis::column](#) *c, [ibis::fileManager::storage](#) *st, uint32_t offset=8)
- **mesa** (const [ibis::column](#) *c=0, const char *f=0)
- virtual const char * **name** () const
Returns the name of the index, similar to the function type, but returns a string instead.
- virtual uint32_t **numBins** () const
- virtual void **print** (std::ostream &out) const
Prints human readable information.
- virtual void **speedTest** (std::ostream &out) const
Time some logical operations and print out their speed.
- virtual INDEX_TYPE **type** () const
Returns an index type identifier.
- virtual float **undecidable** (const [ibis::qContinuousRange](#) &expr, [ibis::bitvector](#) &iffy) const
Mark the position of the rows that can not be decided with this index.

- virtual void `write` (const char *dt) const
Save index to a file.

Protected Member Functions

- virtual double `computeSum` () const

3.62.1 Detailed Description

This class implements the two-side range encoding from Chan and Ioannidis.

3.62.2 Member Function Documentation

3.62.2.1 void `ibis::mesa::estimate` (const `ibis::qContinuousRange` & *expr*, `ibis::bitvector` & *lower*, `ibis::bitvector` & *upper*) const [virtual]

Computes an approximation of hits as a pair of lower and upper bounds.

Parameters:

expr the query expression to be evaluated.

lower a bitvector marking a subset of the hits. All rows marked with one (1) are definitely hits.

upper a bitvector marking a superset of the hits. All hits are marked with one, but some of the rows marked one may not be hits. If the variable upper is empty, the variable lower is assumed to contain the exact answer.

Reimplemented from `ibis::bin`.

3.62.2.2 long `ibis::mesa::evaluate` (const `ibis::qContinuousRange` & *expr*, `ibis::bitvector` & *hits*) const [virtual]

To evaluate the exact hits.

On success, return the number of hits, otherwise a negative value is returned.

Reimplemented from `ibis::bin`.

3.62.2.3 double `ibis::mesa::getSum` () const [virtual]

Compute the approximate sum of all the values indexed.

If it decides that computing the sum directly from the vertical partition is more efficient, it will return NaN immediately.

Reimplemented from `ibis::bin`.

3.62.2.4 void `ibis::mesa::print` (std::ostream & *out*) const [virtual]

Prints human readable information.

Outputs information about the index as text to the specified output stream.

Reimplemented from `ibis::bin`.

3.62.2.5 float `ibis::mesa::undecidable` (const `ibis::qContinuousRange` & *expr*, `ibis::bitvector` & *iffy*) const [virtual]

Mark the position of the rows that can not be decided with this index.

Parameters:

expr the range conditions to be evaluated.

iffy the bitvector marking the positions of rows that can not be decided using the index. Return value is the expected fraction of undecided rows that might satisfy the range conditions.

Reimplemented from [ibis::bin](#).

3.62.2.6 void `ibis::mesa::write` (const char * *dt*) const [virtual]

Save index to a file.

Outputs the index in a compact binary format to the named file or directory. The index file contains a header that can be identified by the function `isIndex`.

Reimplemented from [ibis::bin](#).

The documentation for this class was generated from the following files:

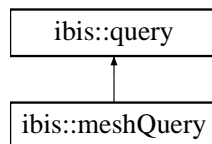
- [ibin.h](#)
- `imesa.cpp`

3.63 `ibis::meshQuery` Class Reference

The class adds more functionality to [ibis::query](#) to handle data from meshes.

```
#include <meshQuery.h>
```

Inheritance diagram for `ibis::meshQuery`:

**Public Member Functions**

- int [getHitsAsRanges](#) (std::vector< std::vector< uint32_t > > ®, const bool merge=false) const
This variant of `getHitsAsRanges` uses the dimensions defined by `ibis::table::getMeshShape()`.
- int [getHitsAsRanges](#) (std::vector< std::vector< uint32_t > > ®, const std::vector< uint32_t > &dim, const bool merge=false) const
Translate hit vector into bounding boxes.
- int [getPointsOnBoundary](#) (std::vector< std::vector< uint32_t > > &bdy) const
The variant of `getPointsOnBoundary` use dimensions returned by `ibis::table::getMeshShape()`.
- int [getPointsOnBoundary](#) (std::vector< std::vector< uint32_t > > &bdy, const std::vector< uint32_t > &dim) const
Determine points with neighbors that are not hits.
- [meshQuery](#) (const char *dir, const `ibis::partList` &tl)
Constructor for recovering from crash.
- [meshQuery](#) (const char *uid, const `part` *et, const char *pref=0)
Constructor for building a new query.

Static Public Member Functions

- static int `bitvectorToCoordinates` (const `ibis::bitvector` &bv, const `std::vector< uint32_t >` &dim, `std::vector< uint32_t >` &coords)

Convert positions in a bit vector to mesh coordinates.

3.63.1 Detailed Description

The class adds more functionality to `ibis::query` to handle data from meshes.

The new functions treats cells of meshes as connected regions in space.

3.63.2 Member Function Documentation

3.63.2.1 `int ibis::meshQuery::bitvectorToCoordinates` (const `ibis::bitvector` &bv, const `std::vector< uint32_t >` &dim, `std::vector< uint32_t >` &coords) `[static]`

Convert positions in a bit vector to mesh coordinates.

It converts the positions of bits that are 1 to coordinates in a regular mesh with deminsions given in `dim`. The C-sytle array ordering is assumed.

3.63.2.2 `int ibis::meshQuery::getHitsAsRanges` (`std::vector< std::vector< uint32_t > >` ®, const bool *merge* = false) const

This variant of `getHitsAsRanges` uses the dimensions defined by `ibis::table::getMeshShape()`.

See also:

[ibis::meshQuery::getHitsAsRanges](#)

[ibis::table::getMeshShape](#)

3.63.2.3 `int ibis::meshQuery::getHitsAsRanges` (`std::vector< std::vector< uint32_t > >` ®, const `std::vector< uint32_t >` &dim, const bool *merge* = false) const

Translate hit vector into bounding boxes.

The bitmap is assumed to be a mapping of a regular grid with dimensions specified in variable `dim`. A row-major ordering (C style multiple dimensional arrays, NOT fortran style) is assumed, that is the slowest varying dimension is `dim[0]`.

Parameters:

reg The return value that contains the list of ranges as hypercubes. Following the convention of typical C/C++ indexing scheme, the lower bounds of the ranges are inclusive and the upper bounds of the ranges are exclusive. For example, a 2D range [2, 3, 5, 10] covers points with coordinates [2, 5], [2, 6], [2, 7], [2, 8] and [2, 9]. This is an example of a line segment with five points. This function may generate hypercubes of any shape or size.

dim The size of the grid. The value `dim.size()` is the dimension of grid. Input argument, nor modified.

merge An optional argument. If true, will attempt to merge line segments generated to form larger hypercubes. Default to false because it make take a significant amount of time to merge the ranges.

This function returns an integer value with the following definition.

- 0 – conversion succeeded (no warning)

- -1 – the bitvector length does not match the product of values in `dim`
- -2 – product of values in `dim` overflows an unsigned integer
- -3 – no hit vector to work with
- -4 – array `dim` is empty

Note:

It can only be called after the functions `estimate` or `evaluate` have been called. The ranges computed after calling `estimate` may be smaller than that computed after calling `evaluate` because `estimate` may not generate the exact answer to the query.

3.63.2.4 `int ibis::meshQuery::getPointsOnBoundary (std::vector< std::vector< uint32_t > > & bdy) const`

The variant of `getPointsOnBoundary` use dimensions returned by `ibis::table::getMeshShape()`.

See also:

[ibis::meshQuery::getPointsOnBoundary](#)
[ibis::table::getMeshShape](#)

3.63.2.5 `int ibis::meshQuery::getPointsOnBoundary (std::vector< std::vector< uint32_t > > & bdy, const std::vector< uint32_t > & dim) const`

Determine points with neighbors that are not hits.

Parameters:

bdy The return value that contains the list of points.

dim The size of the grid. The value `dim.size()` is the dimension of the grid. Each element of `bdy` is a `std::vector` with the same size as `dim`.

See also:

[ibis::meshQuery::getHitsAsRanges](#)

Note:

The inner array has the same number of elements as argument `dim` and the dimensions are ordered the same way as in `dim` as well. All functions in this class assumes that the mesh points are linearized using a raster scan order where `dim[0]` varies the slowest.

The documentation for this class was generated from the following files:

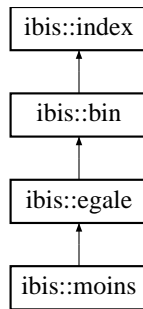
- [meshQuery.h](#)
- [meshQuery.cpp](#)

3.64 `ibis::moins` Class Reference

The multicomponent range code on bins.

```
#include <ibin.h>
```

Inheritance diagram for `ibis::moins`:



Public Member Functions

- long `append` (const `array_t< uint32_t >` &ind)
Append a list of integers representing bin numbers.
- long `append` (const `ibis::moins` &tail)
- virtual long `append` (const char *dt, const char *df, uint32_t nnew)
Extend the index.
- virtual uint32_t `estimate` (const `ibis::qContinuousRange` &expr) const
Returns an upper bound on the number of hits.
- virtual void `estimate` (const `ibis::qContinuousRange` &expr, `ibis::bitvector` &lower, `ibis::bitvector` &upper) const
Computes an approximation of hits as a pair of lower and upper bounds.
- virtual long `evaluate` (const `ibis::qContinuousRange` &expr, `ibis::bitvector` &hits) const
To evaluate the exact hits.
- virtual double `getSum` () const
Compute the approximate sum of all the values indexed.
- `moins` (const `ibis::bin` &rhs, const uint32_t nbase=2)
- `moins` (const `ibis::column` *c, `ibis::fileManager::storage` *st, uint32_t offset=8)
- `moins` (const `ibis::column` *c=0, const char *f=0, const uint32_t nbase=2)
- virtual const char * `name` () const
Returns the name of the index, similar to the function `type`, but returns a string instead.
- virtual void `print` (std::ostream &out) const
Prints human readable information.
- virtual void `speedTest` (std::ostream &out) const
Time some logical operations and print out their speed.
- virtual INDEX_TYPE `type` () const
Returns an index type identifier.
- virtual void `write` (const char *dt) const
Save index to a file.

Protected Member Functions

- virtual double `computeSum () const`

3.64.1 Detailed Description

The multicomponent range code on bins.

Moins is a French word for 'less'.

3.64.2 Member Function Documentation

3.64.2.1 `void ibis::moins::estimate (const ibis::qContinuousRange & expr, ibis::bitvector & lower, ibis::bitvector & upper) const` [virtual]

Computes an approximation of hits as a pair of lower and upper bounds.

Parameters:

expr the query expression to be evaluated.

lower a bitvector marking a subset of the hits. All rows marked with one (1) are definitely hits.

upper a bitvector marking a superset of the hits. All hits are marked with one, but some of the rows marked one may not be hits. If the variable upper is empty, the variable lower is assumed to contain the exact answer.

Reimplemented from [ibis::egale](#).

3.64.2.2 `long ibis::moins::evaluate (const ibis::qContinuousRange & expr, ibis::bitvector & hits) const` [virtual]

To evaluate the exact hits.

On success, return the number of hits, otherwise a negative value is returned.

Reimplemented from [ibis::egale](#).

3.64.2.3 `double ibis::moins::getSum () const` [virtual]

Compute the approximate sum of all the values indexed.

If it decides that computing the sum directly from the vertical partition is more efficient, it will return NaN immediately.

Reimplemented from [ibis::egale](#).

3.64.2.4 `void ibis::moins::print (std::ostream & out) const` [virtual]

Prints human readable information.

Outputs information about the index as text to the specified output stream.

Reimplemented from [ibis::egale](#).

3.64.2.5 `void ibis::moins::write (const char * dt) const` [virtual]

Save index to a file.

Outputs the index in a compact binary format to the named file or directory. The index file contains a header that can be identified by the function `isIndex`.

Reimplemented from [ibis::egale](#).

The documentation for this class was generated from the following files:

- [ibin.h](#)
- `icmoins.cpp`

3.65 `ibis::nameList` Class Reference

A data structure to store a small set of names.

```
#include <util.h>
```

Public Types

- typedef `std::vector< const char * >::const_iterator` **const_iterator**

Public Member Functions

- `const_iterator` **begin** () const
- void **clear** ()
- bool **empty** () const
- `const_iterator` **end** () const
- `size_t` **find** (const char *key) const
Find the order of the key in the list.
- **nameList** (const char *str)
- const char * **operator** * () const
- const char * **operator**[] (size_t i) const
- void **select** (const char *str)
Add more names to the list.
- `size_t` **size** () const

3.65.1 Detailed Description

A data structure to store a small set of names.

The names are sorted in case-insensitive order.

3.65.2 Member Function Documentation

3.65.2.1 `size_t ibis::nameList::find (const char * key) const`

Find the order of the `key` in the list.

If the `key` is in the list it returns the position of the `key`, otherwise it returns the size of the name list.

The documentation for this class was generated from the following files:

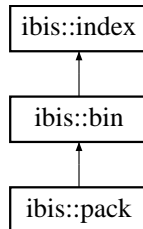
- [util.h](#)
- `query.cpp`

3.66 `ibis::pack` Class Reference

A two-level index.

```
#include <ibin.h>
```

Inheritance diagram for `ibis::pack`:



Public Member Functions

- virtual void **adjustLength** (uint32_t nrows)
- long **append** (const `ibis::pack` &tail)
- virtual long **append** (const char *dt, const char *df, uint32_t nnew)
Extend the index.
- virtual void **binBoundaries** (std::vector< double > &) const
The function `binBoundaries` and `binWeights` return bin boundaries and counts of each bin respectively.
- virtual void **binWeights** (std::vector< uint32_t > &) const
- virtual void **estimate** (const `ibis::qContinuousRange` &expr, `ibis::bitvector` &lower, `ibis::bitvector` &upper) const
Computes an approximation of hits as a pair of lower and upper bounds.
- virtual long **evaluate** (const `ibis::qContinuousRange` &expr, `ibis::bitvector` &hits) const
To evaluate the exact hits.
- virtual double **getSum** () const
Compute the approximate sum of all the values indexed.
- virtual const char * **name** () const
Returns the name of the index, similar to the function `type`, but returns a string instead.
- virtual uint32_t **numBins** () const
- **pack** (const `ibis::bin` &rhs)
- **pack** (const `ibis::column` *c, `ibis::fileManager::storage` *st, uint32_t offset=8)
- virtual void **print** (std::ostream &out) const
Prints human readable information.
- virtual void **read** (`ibis::fileManager::storage` *st)
Reconstructs an index from an array of bytes.
- virtual void **read** (const char *idxfile)
Reconstructs an index from the named file.
- virtual void **speedTest** (std::ostream &out) const
Time some logical operations and print out their speed.

- virtual `INDEX_TYPE type () const`
Returns an index type identifier.
- virtual `float undecidable (const ibis::qContinuousRange &expr, ibis::bitvector &iffy) const`
Mark the position of the rows that can not be decided with this index.
- virtual `void write (const char *dt) const`
Save index to a file.

Protected Member Functions

- virtual `double computeSum () const`
The the approximate sum of all values using the top level bins.

3.66.1 Detailed Description

A two-level index.

Coarse level is cumulative, but not the bottom level.

3.66.2 Member Function Documentation

3.66.2.1 `void ibis::pack::estimate (const ibis::qContinuousRange & expr, ibis::bitvector & lower, ibis::bitvector & upper) const` [virtual]

Computes an approximation of hits as a pair of lower and upper bounds.

Parameters:

expr the query expression to be evaluated.

lower a bitvector marking a subset of the hits. All rows marked with one (1) are definitely hits.

upper a bitvector marking a superset of the hits. All hits are marked with one, but some of the rows marked one may not be hits. If the variable upper is empty, the variable lower is assumed to contain the exact answer.

Reimplemented from `ibis::bin`.

3.66.2.2 `long ibis::pack::evaluate (const ibis::qContinuousRange & expr, ibis::bitvector & hits) const` [virtual]

To evaluate the exact hits.

On success, return the number of hits, otherwise a negative value is returned.

Reimplemented from `ibis::bin`.

3.66.2.3 `double ibis::pack::getSum () const` [virtual]

Compute the approximate sum of all the values indexed.

If it decides that computing the sum directly from the vertical partition is more efficient, it will return NaN immediately.

Reimplemented from `ibis::bin`.

3.66.2.4 void ibis::pack::print (std::ostream & *out*) const [virtual]

Prints human readable information.

Outputs information about the index as text to the specified output stream.

Reimplemented from [ibis::bin](#).

3.66.2.5 void ibis::pack::read (ibis::fileManager::storage * *st*) [virtual]

Reconstructs an index from an array of bytes.

Intended for internal use only!

Reimplemented from [ibis::bin](#).

3.66.2.6 void ibis::pack::read (const char * *idxfile*) [virtual]

Reconstructs an index from the named file.

The name can be the directory containing an index file. In this case, the name of the index file must be the name of the column followed by ".idx" suffix.

Reimplemented from [ibis::bin](#).

3.66.2.7 float ibis::pack::undecidable (const ibis::qContinuousRange & *expr*, ibis::bitvector & *iffy*) const [virtual]

Mark the position of the rows that can not be decided with this index.

Parameters:

expr the range conditions to be evaluated.

iffy the bitvector marking the positions of rows that can not be decided using the index. Return value is the expected fraction of undecided rows that might satisfy the range conditions.

Reimplemented from [ibis::bin](#).

3.66.2.8 void ibis::pack::write (const char * *dt*) const [virtual]

Save index to a file.

Outputs the index in a compact binary format to the named file or directory. The index file contains a header that can be identified by the function `isIndex`.

Reimplemented from [ibis::bin](#).

The documentation for this class was generated from the following files:

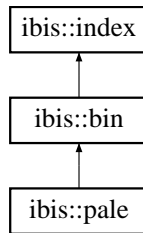
- [ibin.h](#)
- [ixpack.cpp](#)

3.67 **ibis::pale Class Reference**

A two-level index.

```
#include <ibin.h>
```

Inheritance diagram for `ibis::pale`:



Public Member Functions

- virtual void **adjustLength** (uint32_t nrows)
- long **append** (const [ibis::pale](#) &tail)
- virtual long **append** (const char *dt, const char *df, uint32_t nnew)
Extend the index.
- virtual void **binBoundaries** (std::vector< double > &) const
The function binBoundaries and binWeights return bin boundaries and counts of each bin respectively.
- virtual void **binWeights** (std::vector< uint32_t > &) const
- virtual void **estimate** (const [ibis::qContinuousRange](#) &expr, [ibis::bitvector](#) &lower, [ibis::bitvector](#) &upper) const
Computes an approximation of hits as a pair of lower and upper bounds.
- virtual long **evaluate** (const [ibis::qContinuousRange](#) &expr, [ibis::bitvector](#) &hits) const
To evaluate the exact hits.
- virtual const char * **name** () const
Returns the name of the index, similar to the function type, but returns a string instead.
- virtual uint32_t **numBins** () const
- **pale** (const [ibis::bin](#) &rhs)
- **pale** (const [ibis::column](#) *c, [ibis::fileManager::storage](#) *st, uint32_t offset=8)
- virtual void **print** (std::ostream &out) const
Prints human readable information.
- virtual void **read** ([ibis::fileManager::storage](#) *st)
Reconstructs an index from an array of bytes.
- virtual void **read** (const char *idxfile)
Reconstructs an index from the named file.
- virtual void **speedTest** (std::ostream &out) const
Time some logical operations and print out their speed.
- virtual INDEX_TYPE **type** () const
Returns an index type identifier.
- virtual float **undecidable** (const [ibis::qContinuousRange](#) &expr, [ibis::bitvector](#) &iffy) const
Mark the position of the rows that can not be decided with this index.
- virtual void **write** (const char *dt) const
Save index to a file.

3.67.1 Detailed Description

A two-level index.

Coarse level not cumulative, fine level is cumulative.

3.67.2 Member Function Documentation

3.67.2.1 `void ibis::pale::estimate (const ibis::qContinuousRange & expr, ibis::bitvector & lower, ibis::bitvector & upper) const` [virtual]

Computes an approximation of hits as a pair of lower and upper bounds.

Parameters:

expr the query expression to be evaluated.

lower a bitvector marking a subset of the hits. All rows marked with one (1) are definitely hits.

upper a bitvector marking a superset of the hits. All hits are marked with one, but some of the rows marked one may not be hits. If the variable upper is empty, the variable lower is assumed to contain the exact answer.

Reimplemented from [ibis::bin](#).

3.67.2.2 `long ibis::pale::evaluate (const ibis::qContinuousRange & expr, ibis::bitvector & hits) const` [virtual]

To evaluate the exact hits.

On success, return the number of hits, otherwise a negative value is returned.

Reimplemented from [ibis::bin](#).

3.67.2.3 `void ibis::pale::print (std::ostream & out) const` [virtual]

Prints human readable information.

Outputs information about the index as text to the specified output stream.

Reimplemented from [ibis::bin](#).

3.67.2.4 `void ibis::pale::read (ibis::fileManager::storage * st)` [virtual]

Reconstructs an index from an array of bytes.

Intended for internal use only!

Reimplemented from [ibis::bin](#).

3.67.2.5 `void ibis::pale::read (const char * idxfile)` [virtual]

Reconstructs an index from the named file.

The name can be the directory containing an index file. In this case, the name of the index file must be the name of the column followed by ".idx" suffix.

Reimplemented from [ibis::bin](#).

3.67.2.6 `float ibis::pale::undecidable (const ibis::qContinuousRange & expr, ibis::bitvector & iffy) const` [virtual]

Mark the position of the rows that can not be decided with this index.

Parameters:

expr the range conditions to be evaluated.

iffy the bitvector marking the positions of rows that can not be decided using the index. Return value is the expected fraction of undecided rows that might satisfy the range conditions.

Reimplemented from [ibis::bin](#).

3.67.2.7 void `ibis::pale::write` (const char * *dt*) const [virtual]

Save index to a file.

Outputs the index in a compact binary format to the named file or directory. The index file contains a header that can be identified by the function `isIndex`.

Reimplemented from [ibis::bin](#).

The documentation for this class was generated from the following files:

- [ibin.h](#)
- [ixpale.cpp](#)

3.68 `ibis::part` Class Reference

The class [ibis::part](#) represents a partition of a relational table.

```
#include <part.h>
```

Public Types

- typedef std::map< const char *, [column](#) *, [lessi](#) > **columnList**
- enum **TABLE_STATE** {
UNKNOWN_STATE = 0, **STABLE_STATE**, **RECEIVING_STATE**, **PRETRANSITION_STATE**,
TRANSITION_STATE, **POSTTRANSITION_STATE** }

Public Member Functions

- [ibis::fileManager::ACCESS_PREFERENCE](#) **accessHint** (const [ibis::bitvector](#) &mask, unsigned elemsize=4)
const
Evaluate the strategy to access a data file.
- long **append** (const char *dir)
Append data in dir to the current database.
- void **buildIndex** (int nthr=1)
Load and immediately unload indices.
- void **combineNames** ([ibis::table::namesTypes](#) &metalist) const
Update the list of columns with information in this data partition.
- long **commit** (const char *dir)
Commit the active database.
- void **computeMinMax** ()
Go through all the values to compute the min and max for each column.

- long **countHits** (const [ibis::qRange](#) &cmp) const
Count the number of hits for a single range condition.
- const char * **currentDataDir** () const
Return the name of the active data directory.
- const char * **description** () const
Return a text description of the partition.
- void **doBackup** ()
A function to start backing up the active dir.
- template<typename E> long **doScan** (const [array_t](#)< E > &varr, const [ibis::qContinuousRange](#) &cmp, const [ibis::bitvector](#) &mask, [ibis::bitvector](#) &hits) const
- template<typename E> long **doScan** (const [array_t](#)< E > &varr, const [ibis::qRange](#) &cmp, const [ibis::bitvector](#) &mask, [ibis::bitvector](#) &hits) const
Locate the records that satisfy the range condition.
- virtual long **doScan** (const [ibis::compRange](#) &cmp, const [ibis::bitvector](#) &mask, [ibis::bitvector](#) &hits, [ibis::compRange::barrel](#) *bar=0) const
Locate the records that have mark value 1 and satisfy the complex range conditions.
- virtual long **doScan** (const [ibis::compRange](#) &cmp, [ibis::bitvector](#) &hits) const
Locate the records that satisfy the complex range condition.
- virtual long **doScan** (const [ibis::qRange](#) &cmp, const [ibis::bitvector](#) &mask, [ibis::bitvector](#) &hits) const
Evaluate the range condition on the records that are marked 1 in the mask.
- virtual long **doScan** (const [ibis::qRange](#) &cmp, [ibis::bitvector](#) &hits) const
Evaluate the range condition.
- double **estimateCost** (const [ibis::qMultiString](#) &cmp) const
- double **estimateCost** (const [ibis::qString](#) &cmp) const
- double **estimateCost** (const [ibis::qDiscreteRange](#) &cmp) const
- double **estimateCost** (const [ibis::qContinuousRange](#) &cmp) const
Estimate the cost of evaluate the query expression.
- virtual long **estimateMatchAny** (const [ibis::qAnyAny](#) &cmp, [ibis::bitvector](#) &low, [ibis::bitvector](#) &high) const
Estimate a lower bound and an upper bound on the records that are hits.
- long **estimateRange** (const [ibis::qDiscreteRange](#) &cmp, [ibis::bitvector](#) &low, [ibis::bitvector](#) &high) const
Estimate the discrete range condition.
- long **estimateRange** (const [ibis::qContinuousRange](#) &cmp, [ibis::bitvector](#) &low, [ibis::bitvector](#) &high) const
- long **estimateRange** (const [ibis::qDiscreteRange](#) &cmp) const
Return an upper bound on the number of hits.
- long **estimateRange** (const [ibis::qContinuousRange](#) &cmp) const
Return an upper bound on the number of hits.
- int64_t **evaluateJoin** (const std::vector< const [ibis::rangeJoin](#) * > &cmp, const [ibis::bitvector64](#) &trial, [ibis::bitvector64](#) &result) const

Check a set of pairs defined in `trial`.

- `int64_t evaluateJoin` (const `ibis::rangeJoin` &cmp, const `ibis::bitvector64` &trial, `ibis::bitvector64` &result) const

Evaluate all pairs in `trial` to determine whether they really satisfy the range join defined in `cmp`.

- `int64_t evaluateJoin` (const std::vector< const `ibis::rangeJoin` * > &cmp, const `ibis::bitvector` &mask) const
- `int64_t evaluateJoin` (const std::vector< const `ibis::rangeJoin` * > &cmp, const `ibis::bitvector` &mask, `ibis::bitvector64` &pairs) const

Evaluate a join defined with multiple (conjunctive) range join conditions.

- `int64_t evaluateJoin` (const `ibis::rangeJoin` &cmp, const `ibis::bitvector` &mask) const

Return only the number of pairs satisfying the join condition.

- `int64_t evaluateJoin` (const `ibis::rangeJoin` &cmp, const `ibis::bitvector` &mask, const char *pairfile) const

Return the number of pairs satisfying the join condition.

- `int64_t evaluateJoin` (const `ibis::rangeJoin` &cmp, const `ibis::bitvector` &mask, `ibis::bitvector64` &pairs) const

Evaluate a self-join.

- `long evaluateRange` (const `ibis::qDiscreteRange` &cmp, const `ibis::bitvector` &mask, `ibis::bitvector` &res) const

- `long evaluateRange` (const `ibis::qContinuousRange` &cmp, const `ibis::bitvector` &mask, `ibis::bitvector` &res) const

- `long evaluateRIDSet` (const `ibis::RIDSet` &, `ibis::bitvector` &) const

- `long get1DDistribution` (const char *constraints, const char *cname, double begin, double end, double stride, std::vector< size_t > &counts) const

Count the number of records falling in the bins defined by the `begin:end:stride` triplets.

- `long get2DDistribution` (const char *constraints, const char *cname1, double begin1, double end1, double stride1, const char *cname2, double begin2, double end2, double stride2, std::vector< size_t > &counts) const

`ibis::part::get1DDistribution` `ibis::table::getHistogram2D`

- `long get3DDistribution` (const char *constraints, const char *cname1, double begin1, double end1, double stride1, const char *cname2, double begin2, double end2, double stride2, const char *cname3, double begin3, double end3, double stride3, std::vector< size_t > &counts) const

`ibis::part::get1DDistribution` `ibis::table::getHistogram3D`

- `double getActualMax` (const char *name) const

The actual maximum value in the named column.

- `double getActualMin` (const char *name) const

The actual minimum value in the named column.

- `column * getColumn` (uint32_t ind) const

Returns the pointer to the *i*th column.

- `column * getColumn` (const char *name) const

Given a name, return the associated column.

- `double getColumnSum` (const char *name) const

Sum of all value in the named column.

- long [getCumulativeDistribution](#) (const char *constraints, const char *name, uint32_t nbc, double *bounds, uint32_t *counts) const
Compute the conditional distribution and return the distribution in the arrays provided.
- long [getCumulativeDistribution](#) (const char *name, uint32_t nbc, double *bounds, uint32_t *counts) const
This version of [getCumulativeDistribution](#) uses two user supplied arrays `bounds` and `counts`.
- long [getCumulativeDistribution](#) (const char *constraints, const char *name, std::vector< double > &bounds, std::vector< uint32_t > &counts) const
Compute the cumulative distribution of the variable named `name` under the specified constraints.
- long [getCumulativeDistribution](#) (const char *name, std::vector< double > &bounds, std::vector< uint32_t > &counts) const
Compute a cumulative distribution (a cumulative histogram).
- long [getDistribution](#) (const char *name, const char *constraints, uint32_t nbc, double *bounds, uint32_t *counts) const
Compute the conditional binned data distribution with the specified maximum number of bins.
- long [getDistribution](#) (const char *name, uint32_t nbc, double *bounds, uint32_t *counts) const
Compute the binned distribution with the specified maximum number of bins.
- long [getDistribution](#) (const char *constraints, const char *name, std::vector< double > &bounds, std::vector< uint32_t > &counts) const
Compute the conditional binned data distribution.
- long [getDistribution](#) (const char *name, std::vector< double > &bounds, std::vector< uint32_t > &counts) const
Compute the binned distribution of the name variable.
- **info** * [getInfo](#) () const
Return descriptive information about the data partition.
- long [getJointDistribution](#) (const char *constraints, const char *name1, const char *name2, std::vector< double > &bounds1, std::vector< double > &bounds2, std::vector< uint32_t > &counts) const
Compute the joint distribution of two variables.
- const std::vector< std::string > & [getMeshDimensions](#) () const
Return the name of the dimensions corresponding to the vector returned from [getMeshShape](#).
- const std::vector< uint32_t > & [getMeshShape](#) () const
In many scientific applications, data are defined on meshes.
- const char * [getMetaTag](#) (const char *) const
Return the value of the meta tag with the specified name.
- [array_t< rid_t >](#) * [getRIDs](#) (const [ibis::bitvector](#) &mask) const
- [array_t< rid_t >](#) * [getRIDs](#) () const
- uint32_t [getRowNumber](#) (const [rid_t](#) &rid) const
Return the row number of the row with specified RID.
- TABLE_STATE [getState](#) () const
Return the current state of data partition.

- TABLE_STATE **getStateNoLocking** () const
- float **getUndecidable** (const ibis::qDiscreteRange &cmp, **ibis::bitvector** &iffy) const
Discover the records that can not be decided using the index.
- float **getUndecidable** (const **ibis::qContinuousRange** &cmp, **ibis::bitvector** &iffy) const
Discover the records that can not be decided using the index.
- bool **hasRIDs** () const
- void **indexSpec** (const char *)
Replace existing index specification with a new one.
- const char * **indexSpec** () const
Return the current index specification.
- void **loadIndex** (const char *opt=0) const
Load all indices.
- void **logMessage** (const char *event, const char *fmt,...) const
- void **logWarning** (const char *event, const char *fmt,...) const
- long **lookforString** (const ibis::qMultiString &cmp) const
- long **lookforString** (const **ibis::qString** &cmp) const
Return an upper bound of the number of records that have the exact string value.
- long **lookforString** (const ibis::qMultiString &cmp, **ibis::bitvector** &low) const
Determine the records that have the exact string values.
- long **lookforString** (const **ibis::qString** &cmp, **ibis::bitvector** &low) const
Find all records that has the exact string value.
- virtual long **matchAny** (const **ibis::qAnyAny** &cmp, const **ibis::bitvector** &mask, **ibis::bitvector** &hits) const
Perform exact match operation for an AnyAny query.
- virtual long **matchAny** (const **ibis::qAnyAny** &cmp, **ibis::bitvector** &hits) const
- bool **matchMetaTags** (const ibis::resource::vList &mtags) const
Match multiple name-value pairs.
- bool **matchMetaTags** (const std::vector< const char * > &mtags) const
Match multiple name-value pairs against the internally stored meta tags.
- bool **matchNameValuePair** (const char *name, const char *value) const
Match a name-value pair in the meta tags.
- std::string **metaTags** () const
Return the list of meta tags as a single string.
- const char * **name** () const
Return the name of the partition.
- size_t **nColumns** () const
Return the number of attributes in the partition.
- virtual long **negativeScan** (const **ibis::qRange** &cmp, const **ibis::bitvector** &mask, **ibis::bitvector** &hits) const

Compute the records (marked 1 in the mask) that does not satisfy the range condition.

- `size_t nRows () const`
Return the number of rows.
- `part (const ibis::resource::vList &mtags)`
- `part (const std::vector< const char * > &mtags)`
- `part (const char *adir, const char *bdir)`
Initialize a table from the named directories.
- `part (const char *prefix=0)`
Initialize a table object. Use `gParameters` to get the file names.
- `void print (std::ostream &out) const`
Output a description of every column of the data partition.
- `void purgeIndexFiles () const`
Remove existing index files!
- `void queryTest (const char *pref, long *nerrors) const`
- `void quickTest (const char *pref, long *nerrors) const`
- `long reorder (const ibis::table::stringList &names)`
- `long reorder ()`
Reorder all columns of a partition.
- `long rollback ()`
Rollback(revert) to previous data set.
- `array_t< double > * selectDoubles (const char *name, const ibis::bitvector &mask) const`
- `array_t< float > * selectFloats (const char *name, const ibis::bitvector &mask) const`
- `array_t< int32_t > * selectInts (const char *name, const ibis::bitvector &mask) const`
- `array_t< int64_t > * selectLongs (const char *name, const ibis::bitvector &mask) const`
- `array_t< uint32_t > * selectUInts (const char *name, const ibis::bitvector &mask) const`
- `virtual long selfTest (int nth=1, const char *pref=0) const`
Perform predefined set of tests and return the number of failures.
- `void setMeshShape (const char *shape)`
Digest the mesh shape stored in the string.
- `time_t timestamp () const`
Return the time stamp on the partition.
- `void unloadIndex () const`
Unload index.
- `void updateTDC () const`
Write the TDC file to record the changes to the partition.

Static Public Member Functions

- static uint32_t **countPages** (const [ibis::bitvector](#) &mask, unsigned elemsize=4)
Given a [bitvector](#), compute the number of pages would be accessed.
- static void **genName** (const [ibis::resource::vList](#) &mtags, std::string &name)
Generate name for a partition based on the meta tags.
- static void **genName** (const std::vector< const char * > &mtags, std::string &name)
Generate name for a partition based on the meta tags.
- static char * **readMetaTags** (const char *const dir)
A class function to read the meta tags in the tdc file.

Protected Member Functions

- long **append1** (const char *dir)
- long **append2** (const char *dir)
- long **appendToBackup** (const char *dir)
Append data in dir to the partition in the backup directory.
- template<typename T1, typename T2> long **count2DBins** (const [array_t](#)< T1 > &vals1, const double &begin1, const double &end1, const double &stride1, const [array_t](#)< T2 > &vals2, const double &begin2, const double &end2, const double &stride2, std::vector< size_t > &counts) const
Count the number of values falling in 2D bins.
- template<typename T1, typename T2, typename T3> long **count3DBins** (const [array_t](#)< T1 > &vals1, const double &begin1, const double &end1, const double &stride1, const [array_t](#)< T2 > &vals2, const double &begin2, const double &end2, const double &stride2, const [array_t](#)< T3 > &vals3, const double &begin3, const double &end3, const double &stride3, std::vector< size_t > &counts) const
Count the number of values falling in 3D bins.
- void **deriveBackupDirName** ()
- void **digestMeshShape** (const char *shape)
Convert the string describing the shape into internal storage format.
- void **doByteCompare** (const char *file, const [ibis::bitvector](#) &mask, [ibis::bitvector](#) &hits, const [ibis::qRange](#) &cmp) const
- template<typename T, typename F1, typename F2> long **doCompare** (const [array_t](#)< T > &vals, F1 cmp1, F2 cmp2, const [ibis::bitvector](#) &mask, [ibis::bitvector](#) &hits) const
The actual scan function.
- template<typename T, typename F> long **doCompare** (const [array_t](#)< T > &vals, F cmp, const [ibis::bitvector](#) &mask, [ibis::bitvector](#) &hits) const
Accepts an externally passed comparison operator.
- template<typename T> void **doCompare** (const char *file, const [ibis::bitvector](#) &mask, [ibis::bitvector](#) &hits, const [ibis::qRange](#) &cmp) const
- template<typename T> void **doCompare** (const [array_t](#)< T > &array, const [ibis::bitvector](#) &mask, [ibis::bitvector](#) &hits, const [ibis::qRange](#) &cmp) const
- void **doCompare** (const [array_t](#)< double > &array, const [ibis::bitvector](#) &mask, [ibis::bitvector](#) &hits, const [ibis::qRange](#) &cmp) const

- void **doCompare** (const [array_t](#)< float > &array, const [ibis::bitvector](#) &mask, [ibis::bitvector](#) &hits, const [ibis::qRange](#) &cmp) const
- void **doCompare** (const [array_t](#)< uint32_t > &array, const [ibis::bitvector](#) &mask, [ibis::bitvector](#) &hits, const [ibis::qRange](#) &cmp) const
- void **doCompare** (const [array_t](#)< int32_t > &array, const [ibis::bitvector](#) &mask, [ibis::bitvector](#) &hits, const [ibis::qRange](#) &cmp) const
- template<typename T, typename F1, typename F2> long **doCompare0** (const [array_t](#)< T > &vals, F1 cmp1, F2 cmp2, const [ibis::bitvector](#) &mask, [ibis::bitvector](#) &hits) const
This version uses uncompressed bitvector to store the scan results internally.
- template<typename T, typename F> long **doCompare0** (const [array_t](#)< T > &vals, F cmp, const [ibis::bitvector](#) &mask, [ibis::bitvector](#) &hits) const
This version uses an uncompressed bitvector to store the scan results internally.
- template<typename T, typename F1, typename F2> long **doCount** (const [array_t](#)< T > &vals, const [ibis::bitvector](#) &mask, F1 cmp1, F2 cmp2) const
- template<typename T, typename F> long **doCount** (const [array_t](#)< T > &vals, const [ibis::bitvector](#) &mask, F cmp) const
- template<typename T> long **doCount** (const [array_t](#)< T > &vals, const [ibis::qRange](#) &cmp, const [ibis::bitvector](#) &mask) const
- template<typename T> long **doCount** (const [ibis::qRange](#) &cmp) const
- void **doDoubleCompare** (const char *file, const [ibis::bitvector](#) &mask, [ibis::bitvector](#) &hits, const [ibis::qRange](#) &cmp) const
- void **doFloatCompare** (const char *file, const [ibis::bitvector](#) &mask, [ibis::bitvector](#) &hits, const [ibis::qRange](#) &cmp) const
- void **doIntCompare** (const char *file, const [ibis::bitvector](#) &mask, [ibis::bitvector](#) &hits, const [ibis::qRange](#) &cmp) const
- void **doShortCompare** (const char *file, const [ibis::bitvector](#) &mask, [ibis::bitvector](#) &hits, const [ibis::qRange](#) &cmp) const
- void **doUByteCompare** (const char *file, const [ibis::bitvector](#) &mask, [ibis::bitvector](#) &hits, const [ibis::qRange](#) &cmp) const
- void **doUIntCompare** (const char *file, const [ibis::bitvector](#) &mask, [ibis::bitvector](#) &hits, const [ibis::qRange](#) &cmp) const
- void **doUShortCompare** (const char *file, const [ibis::bitvector](#) &mask, [ibis::bitvector](#) &hits, const [ibis::qRange](#) &cmp) const
- void **extendMetaTags** ()
- void **freeRIDs** () const
Remove the rids list from memory.
- void **logError** (const char *event, const char *fmt,...) const
Log functions.
- void **makeBackupCopy** ()
- template<typename T> void **negativeCompare** (const char *file, const [ibis::bitvector](#) &mask, [ibis::bitvector](#) &hits, const [ibis::qRange](#) &cmp) const
- template<typename T> void **negativeCompare** (const [array_t](#)< T > &array, const [ibis::bitvector](#) &mask, [ibis::bitvector](#) &hits, const [ibis::qRange](#) &cmp) const
- void **negativeCompare** (const [array_t](#)< double > &array, const [ibis::bitvector](#) &mask, [ibis::bitvector](#) &hits, const [ibis::qRange](#) &cmp) const
- void **negativeCompare** (const [array_t](#)< float > &array, const [ibis::bitvector](#) &mask, [ibis::bitvector](#) &hits, const [ibis::qRange](#) &cmp) const
- void **negativeCompare** (const [array_t](#)< uint32_t > &array, const [ibis::bitvector](#) &mask, [ibis::bitvector](#) &hits, const [ibis::qRange](#) &cmp) const
- void **negativeCompare** (const [array_t](#)< int32_t > &array, const [ibis::bitvector](#) &mask, [ibis::bitvector](#) &hits, const [ibis::qRange](#) &cmp) const

- void **negativeDoubleCompare** (const char *file, const [ibis::bitvector](#) &mask, [ibis::bitvector](#) &hits, const [ibis::q-Range](#) &cmp) const
- void **negativeFloatCompare** (const char *file, const [ibis::bitvector](#) &mask, [ibis::bitvector](#) &hits, const [ibis::q-Range](#) &cmp) const
- void **negativeIntCompare** (const char *file, const [ibis::bitvector](#) &mask, [ibis::bitvector](#) &hits, const [ibis::q-Range](#) &cmp) const
- void **negativeUIntCompare** (const char *file, const [ibis::bitvector](#) &mask, [ibis::bitvector](#) &hits, const [ibis::q-Range](#) &cmp) const
- long **packCumulativeDistribution** (const std::vector< double > &bounds, const std::vector< uint32_t > &counts, uint32_t nbc, double *bptr, uint32_t *cptr) const
Pack a cumulative distribution stored in two std::vectors into two arrays provided by the caller.
- long **packDistribution** (const std::vector< double > &bounds, const std::vector< uint32_t > &counts, uint32_t nbc, double *bptr, uint32_t *cptr) const
Pack a binned distribution.
- void **readMeshShape** (const char *const dir)
Read shape of the mesh from tdc file.
- void **readRIDs** () const
Read RIDs from file 'rids'.
- int **readTDC** (size_t &nrows, columnList &plist, const char *dir)
Read TDC file.
- template<typename T> long **reorderValues** (const char *fname, const [array_t](#)< uint32_t > &indin, [array_t](#)< uint32_t > &indout, [array_t](#)< uint32_t > &starts)
Write the named data file in a segmented sorted order.
- void **setMetaTags** (const std::vector< const char * > &mts)
Make a deep copy of the incoming name-value pairs.
- void **setMetaTags** (const [ibis::resource::vList](#) &mts)
Make a deep copy of the incoming name-value pairs.
- long **verifyBackupDir** ()
- void **writeTDC** (const uint32_t nrows, const columnList &plist, const char *dir) const
Write TDC file.
- template<typename T> long **writeValues** (const char *fname, const [array_t](#)< uint32_t > &ind)
Write the named data file with values in the given order.

Protected Attributes

- char * **activeDir**
The active data directory.
- char * **backupDir**
The backup data directory.
- std::vector< const [column](#) * > **colorder**
An ordering of columns.

- `columnList` [columns](#)
List of the columns.
- `char *` [idxstr](#)
Index specification.
- `char *` [m_desc](#)
Free form description of the partition.
- `char *` [m_name](#)
Name of the data partition.
- `ibis::resource::vList` [metaList](#)
Meta tags as name-value pairs.
- `ibis::part::cleaner *` [myCleaner](#)
The cleaner for the file manager.
- `size_t` [nEvents](#)
Number of events (rows) in the partition.
- `array_t< rid_t > *` [rids](#)
The object IDs (row id).
- `std::vector< std::string >` [shapeName](#)
Names of the dimensions.
- `std::vector< uint32_t >` [shapeSize](#)
Sizes of the dimensions.
- `TABLE_STATE` [state](#)
- `time_t` [switchTime](#)
Time of last switch operation.

Friends

- class [advisoryLock](#)
- class [cleaner](#)
- struct [info](#)
- class [mutexLock](#)
- class [readLock](#)
- class [writeLock](#)

Classes

- class [advisoryLock](#)
An non-blocking version of [writeLock](#).
- class [barrel](#)
To read a list of variables at the same time.

- class [cleaner](#)
A cleaner to be used by the `fileManager::unload` function.
- struct **indexBuilderPool**
- struct [info](#)
A simple class to describe an `ibis::part` object.
- class [mutexLock](#)
Provide a mutual exclusion lock on an `ibis::part` object.
- class [readLock](#)
Provide a read lock on an `ibis::part`.
- struct **thrArg**
- class [vault](#)
To read variables in certain order.
- class [writeLock](#)
Provide a write lock on an `ibis::part`.

3.68.1 Detailed Description

The class `ibis::part` represents a partition of a relational table.

The current implementation is designed to work with vertically partitioned data files. This class contains common information and operations on a partition.

3.68.2 Constructor & Destructor Documentation

3.68.2.1 `ibis::part::part (const char * adir, const char * bdir)`

Initialize a table from the named directories.

Must have full and complete path.

3.68.3 Member Function Documentation

3.68.3.1 `long ibis::part::append (const char * dir)`

Append data in `dir` to the current database.

Return the number of rows actually added. It is possible to rollback the append operation before commit.

3.68.3.2 `long ibis::part::appendToBackup (const char * dir)` [protected]

Append data in `dir` to the partition in the backup directory.

Return the number of rows actually appended.

3.68.3.3 `long ibis::part::commit (const char * dir)`

Commit the active database.

No longer able to rollback afterward. Return the number of records committed.

3.68.3.4 `template<typename T, typename F1, typename F2> long ibis::part::doCompare (const array_t< T > & vals, F1 cmp1, F2 cmp2, const ibis::bitvector & mask, ibis::bitvector & hits) const` `[protected]`

The actual scan function.

This one chooses whether the internal bitvector for storing the scan results will be compressed or not. It always returns a compressed bitvector.

3.68.3.5 `template<typename T, typename F> long ibis::part::doCompare (const array_t< T > & vals, F cmp, const ibis::bitvector & mask, ibis::bitvector & hits) const` `[protected]`

Accepts an externally passed comparison operator.

It chooses whether the bitvector `hits` will be compressed internally based on the number of set bits in the `mask`.

3.68.3.6 `template<typename E> long ibis::part::doScan (const array_t< E > & varr, const ibis::qRange & cmp, const ibis::bitvector & mask, ibis::bitvector & hits) const`

Locate the records that satisfy the range condition.

Since the values are provided, this function does not check the name of the variable involved in the range condition.

3.68.3.7 `long ibis::part::doScan (const ibis::qRange & cmp, ibis::bitvector & hits) const` `[virtual]`

Evaluate the range condition.

Scan the base data to resolve the range condition.

3.68.3.8 `long ibis::part::estimateMatchAny (const ibis::qAnyAny & cmp, ibis::bitvector & low, ibis::bitvector & high) const` `[virtual]`

Estimate a lower bound and an upper bound on the records that are hits.

The bitvector `low` contains records that are hits (for sure) and the bitvector `high` contains records that are possible hits.

3.68.3.9 `int64_t ibis::part::evaluateJoin (const std::vector< const ibis::rangeJoin * > & cmp, const ibis::bitvector64 & trial, ibis::bitvector64 & result) const`

Check a set of pairs defined in `trial`.

This version works on multiple (conjunctive) join conditions.

3.68.3.10 `int64_t ibis::part::evaluateJoin (const ibis::rangeJoin & cmp, const ibis::bitvector64 & trial, ibis::bitvector64 & result) const`

Evaluate all pairs in `trial` to determine whether they really satisfy the range join defined in `cmp`.

The result is stored in the argument `result`. This function returns the number of hits found.

3.68.3.11 `int64_t ibis::part::evaluateJoin (const ibis::rangeJoin & cmp, const ibis::bitvector & mask, const char * pairfile) const` `[inline]`

Return the number of pairs satisfying the join condition.

In addition, write the pairs into the file named `pairfile`.

3.68.3.12 `int64_t ibis::part::evaluateJoin (const ibis::rangeJoin & cmp, const ibis::bitvector & mask, ibis::bitvector64 & pairs) const` `[inline]`

Evaluate a self-join.

Return the number of pairs satisfying join condition. Only records marked with `mask=1` are considered. The result pairs are stored in the bitvector `pairs`. A pair $\langle i, j \rangle$ would be marked at position $i * nRows() + j$ in `pairs`.

3.68.3.13 `long ibis::part::get1DDistribution (const char * constraints, const char * cname, double begin, double end, double stride, std::vector< size_t > & counts) const`

Count the number of records falling in the bins defined by the `begin:end:stride` triplets.

The triplets defines

```
std::ceil((end-begin)/stride)
```

bins. When this function completes successfully, it the array `counts` should

```
std::ceil((end-begin)/stride)
```

values, one for each bin. The return value should be the number of bins, otherwise, it indicates an error. If array `counts` has the same size as the number of bins on input, the count values will be added to the array. This is intended to be used to accumulate counts from different data partitions. If the array `counts` does not have the correct size, it will be resized to the correct size and initialized to contain only zero before counting the the current data partition.

[ibis::table::getHistogram](#)

3.68.3.14 `ibis::column * ibis::part::getColumn (const char * name) const [inline]`

Given a name, return the associated column.

Return nil pointer if the name is not found.

3.68.3.15 `long ibis::part::getCumulativeDistribution (const char * constraints, const char * name, uint32_t nbc, double * bounds, uint32_t * counts) const`

Compute the conditional distribution and return the distribution in the arrays provided.

The minimum number of bins expected is four (4). This function will return error code -1 if the value of `nbc` is less than 4.

3.68.3.16 `long ibis::part::getCumulativeDistribution (const char * name, uint32_t nbc, double * bounds, uint32_t * counts) const`

This version of `getCumulativeDistribution` uses two user supplied arrays `bounds` and `counts`.

The actual number of elements filled by this function is the return value, which is guaranteed to be no larger than the input value of `nbc`.

3.68.3.17 `long ibis::part::getCumulativeDistribution (const char * constraints, const char * name, std::vector< double > & bounds, std::vector< uint32_t > & counts) const`

Compute the cumulative distribution of the variable named `name` under the specified constraints.

Note:

The constraints have the same syntax as the where-clause of the queries. Here are two example, "a < 5 and 3.5 >= b >= 1.9" and "a * a + b * b > 55 and sqrt(c) > 2."

3.68.3.18 `long ibis::part::getCumulativeDistribution (const char * name, std::vector< double > & bounds, std::vector< uint32_t > & counts) const`

Compute a cumulative distribution (a cumulative histogram).

It returns the number of entries in arrays `bounds` and `counts`. The content of `counts[i]` will be the number of records in the named column that are less than `bounds[i]`. The last element in array `bounds` is larger than returned by function `getColumnMax`.

3.68.3.19 `long ibis::part::getDistribution (const char * constraints, const char * name, uint32_t nbc, double * bounds, uint32_t * counts) const`

Compute the conditional binned data distribution with the specified maximum number of bins.

The minimum number of bins expected is four (4). This function will return error code -1 if the value of `nbc` is less than 4.

3.68.3.20 `long ibis::part::getDistribution (const char * name, uint32_t nbc, double * bounds, uint32_t * counts) const`

Compute the binned distribution with the specified maximum number of bins.

The minimum number of bins expected is four (4). This function will return error code -1 if the value of `nbc` is less than 4.

3.68.3.21 `long ibis::part::getDistribution (const char * constraints, const char * name, std::vector< double > & bounds, std::vector< uint32_t > & counts) const`

Compute the conditional binned data distribution.

If the input array `bounds` contains distinct values in ascending order, the array will be used as bin boundaries. Otherwise, the bin boundaries are automatically determined by this function. The basic rule for determining the number of bins is that if there are less than 10,000 distinct values, than everyone value is counted separately, otherwise 1000 bins will be used and each bin will contain roughly the same number of records.

3.68.3.22 `long ibis::part::getDistribution (const char * name, std::vector< double > & bounds, std::vector< uint32_t > & counts) const`

Compute the binned distribution of the name variable.

The array `bounds` defines the following bins:

(..., `bounds[0]`) [`bounds[0]`, `bounds[1]`] ... [`bounds.back()`, ...).

In other word, `bounds[n]` defines (n+1) bins, with two open bins at the two ends. The array `counts` contains the number of rows fall into each bin. On a successful return from this function, the return value of this function is the number of bins defined, which is the same as the size of array `counts` but one larger than the size of array `bounds`.

3.68.3.23 `long ibis::part::getJointDistribution (const char * constraints, const char * name1, const char * name2, std::vector< double > & bounds1, std::vector< double > & bounds2, std::vector< uint32_t > & counts) const`

Compute the joint distribution of two variables.

It returns three arrays, `bounds1`, `bounds2`, and `counts`. The arrays `bounds1` and `bounds2` defines two sets of bins one for each variable. Together they define

```
(bounds1.size()+1)
(bounds2.size()+1)
```

bins for the 2-D joint distributions. The array `counts` contains a count for each of the bins. On successful completion of this function, it return the number of bins.

3.68.3.24 `const std::vector<uint32_t>& ibis::part::getMeshShape () const` `[inline]`

In many scientific applications, data are defined on meshes.

The following functions assumes the meshes are regular. Under this assumption, each column can be viewed as a multi-dimensional array, such as `A[nz][ny][nx]`. Following the convention in `C/C++`. The dimensions of the array are ordered from left to the right, with the left most being the slowest varying dimension and the right most being the fast varying dimension. This assumption about the dimensions is explicitly used in `query.cpp` in functions `toRanges`, `range2d`, `range3d` and `rangend`. The function `getMeshShape` returns the sizes of the dimensions in a vector.

3.68.3.25 `float ibis::part::getUndecidable (const ibis::qContinuousRange & cmp, ibis::bitvector & iffy) const`

Discover the records that can not be decided using the index.

Logically, `iffy = high - low`, were `high` and `low` are computed from `estimateRange`. The return value is the estimated fraction of records that might satisfy the range condition.

3.68.3.26 `void ibis::part::loadIndex (const char * opt = 0) const`

Load all indices.

If none existed, build them. If an index exists already, it reads metadata about the index into memory. Newly built indices are completely in memory. An index can not be built correctly if it is too large to fit in memory!

3.68.3.27 `long ibis::part::lookforString (const ibis::qMultiString & cmp, ibis::bitvector & low) const`

Determine the records that have the exact string values.

Actual work done in the function search of the string-valued column. It produces no hit if the name is not a string-valued column.

3.68.3.28 `long ibis::part::matchAny (const ibis::qAnyAny & cmp, const ibis::bitvector & mask, ibis::bitvector & hits) const` `[virtual]`

Perform exact match operation for an `AnyAny` query.

The bulk of the work is performed as range query.

3.68.3.29 `std::string ibis::part::metaTags () const`

Return the list of meta tags as a single string.

The meta tags appears as `'name=value'` pairs separated by comma (,).

3.68.3.30 `void ibis::part::purgeIndexFiles () const`

Remove existing index files!

This function is useful after changing the index specification before rebuilding a set of new indices.

3.68.3.31 `int ibis::part::readTDC (size_t & nrows, columnList & plist, const char * dir)` `[protected]`

Read TDC file.

If `dir` is the `activeDir`, it will also update the content of `*this`, otherwise it will only modify arguments `nrows` and `plist`. If this function completes successfully, it returns the maximum length of the column names. Otherwise, it returns a value of zero or less to indicate errors.

Remarks:

The metadata file is named `"-part.txt"`. It is a plain text file with a fixed structure. The prefix `'-'` is to ensure that none of the data files could possibly have the same name (since `'-'` can't appear in any column names). This file was previously named `"table.tdc"` and this function still recognize this old name.

3.68.3.32 `long ibis::part::reorder ()`

Reorder all columns of a partition.

The lowest cardinality column is ordered first. Only integral valued columns are used in sorting. Returns the number of rows reordered when successful, otherwise return a negative number and the base data is corrupt! Danger: This function does not update null masks!

3.68.3.33 `long ibis::part::rollback ()`

Rollback(revert) to previous data set.

Can only undo the last change to the database, currently only function `append` can change the database.

3.68.3.34 `void ibis::part::setMeshShape (const char * shape) [inline]`

Digest the mesh shape stored in the string.

The shape can be just numbers, e.g., `"(10, 12, 14)"`, or `'name=value'` pairs, e.g., `"(nz=10, ny=12, nx=14)"`.

3.68.3.35 `void ibis::part::setMetaTags (const std::vector< const char * > & mts) [protected]`

Make a deep copy of the incoming name-value pairs.

The even element is assumed to be the name and the odd element is assumed to be the value. If the last name is not followed by a value it is assumed to have the value of `'*'`, which indicates do-not-care.

The documentation for this class was generated from the following files:

- [part.h](#)
- [part.cpp](#)
- [parti.cpp](#)
- [party.cpp](#)

3.69 `ibis::part::advisoryLock` Class Reference

An non-blocking version of [writeLock](#).

```
#include <part.h>
```

Public Member Functions

- `bool acquired () const`
- `advisoryLock (const part *tbl, const char *m)`

3.69.1 Detailed Description

An non-blocking version of `writeLock`.

The function `acquired` returns true is the object has acquired a lock successfully, otherwise the function returns false.

The documentation for this class was generated from the following file:

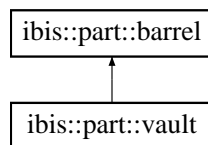
- [part.h](#)

3.70 `ibis::part::barrel` Class Reference

To read a list of variables at the same time.

```
#include <part.h>
```

Inheritance diagram for `ibis::part::barrel`:



Public Member Functions

- `barrel` (const `ibis::part *t=0`)
- virtual long `close` ()
Close all open files.
- const `ibis::column * getColumn` (uint32_t i) const
- void `getNullMask` (`ibis::bitvector &mask`) const
- virtual long `open` (const `ibis::part *t=0`)
Open all data files.
- virtual long `read` ()
Read one value for each variable.
- virtual long `seek` (uint32_t pos)
Move the file pointers to the posth record.
- uint32_t `tell` () const
- virtual `~barrel` ()
Destructor closes the open files.

Protected Attributes

- const `ibis::part * _tbl`
- `std::vector< const ibis::column * > cols`
- `std::vector< int > fdes`
- uint32_t `position`
- `std::vector< ibis::fileManager::storage * > stores`

3.70.1 Detailed Description

To read a list of variables at the same time.

This implementation opens each data file and read the values from the files one at a time.

3.70.2 Member Function Documentation

3.70.2.1 `long ibis::part::barrel::read ()` [virtual]

Read one value for each variable.

All values are internally stored as doubles.

Reimplemented in [ibis::part::vault](#).

3.70.2.2 `long ibis::part::barrel::seek (uint32_t pos)` [virtual]

Move the file pointers to the posth record.

Return 0 for success and other integer for error.

Reimplemented in [ibis::part::vault](#).

The documentation for this class was generated from the following files:

- [part.h](#)
- [part.cpp](#)

3.71 `ibis::part::cleaner` Class Reference

A cleaner to be used by the `fileManager::unload` function.

```
#include <part.h>
```

Public Member Functions

- `cleaner` (const [part](#) *tbl)
- virtual void `operator() ()` const

3.71.1 Detailed Description

A cleaner to be used by the `fileManager::unload` function.

The documentation for this class was generated from the following file:

- [part.h](#)

3.72 `ibis::part::info` Struct Reference

A simple class to describe an [ibis::part](#) object.

```
#include <part.h>
```

Public Member Functions

- `info` (const [ibis::part](#) &tbl)
- `info` (const char *na, const char *de, const uint64_t &nr, const [ibis::part::columnList](#) &co)

Public Attributes

- `std::vector< ibis::column::info * >` `cols`
The list of columns in the partition.
- `const char *` `description`
A free-form description of the partition.
- `const char *` `metaTags`
A string of name-value pairs.
- `const char *` `name`
Partition name.
- `const uint64_t` `nrows`
The number of rows in the partition.

3.72.1 Detailed Description

A simple class to describe an `ibis::part` object.

All members are public and read-only. An info object can not last longer than the `ibis::part` object used to create it.

The documentation for this struct was generated from the following files:

- `part.h`
- `part.cpp`

3.73 `ibis::part::mutexLock` Class Reference

Provide a mutual exclusion lock on an `ibis::part` object.

```
#include <part.h>
```

Public Member Functions

- `mutexLock` (`const part *tbl`, `const char *m`)

3.73.1 Detailed Description

Provide a mutual exclusion lock on an `ibis::part` object.

The documentation for this class was generated from the following file:

- `part.h`

3.74 `ibis::part::readLock` Class Reference

Provide a read lock on an `ibis::part`.

```
#include <part.h>
```

Public Member Functions

- `readLock` (const `part` *tbl, const char *m)

3.74.1 Detailed Description

Provide a read lock on an `ibis::part`.

Routines need read access to `ibis::part` class should use this class instead directly calling `ibis::partgainReadAccess` so that in case of exceptions, the release command would be always called.

The documentation for this class was generated from the following file:

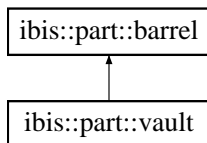
- [part.h](#)

3.75 `ibis::part::vault` Class Reference

To read variables in certain order.

```
#include <part.h>
```

Inheritance diagram for `ibis::part::vault`:



Public Member Functions

- virtual long `open` (const `ibis::part` *t=0)
Open all data files.
- virtual long `read` ()
Read the values at the current position.
- long `seek` (double val)
Move to the first position that value(var) >= val.
- virtual long `seek` (uint32_t pos)
Move the logical position.
- uint32_t `tellReal` () const
Tell the physical record number.
- `vault` (const `ibis::roster` &r)

3.75.1 Detailed Description

To read variables in certain order.

A version of barrel that keys on an index array (i.e., a roster).

3.75.2 Member Function Documentation

3.75.2.1 `long ibis::part::vault::read () [virtual]`

Read the values at the current position.

Treat `position` as the logical position, the physical position is `_roster[position]`.

Reimplemented from [ibis::part::barrel](#).

3.75.2.2 `uint32_t ibis::part::vault::tellReal () const`

Tell the physical record number.

User may called `_roster[tell()]` to avoid the overhead of calling this function.

The documentation for this class was generated from the following files:

- [part.h](#)
- [part.cpp](#)

3.76 `ibis::part::writeLock` Class Reference

Provide a write lock on an [ibis::part](#).

```
#include <part.h>
```

Public Member Functions

- `writeLock` (const [part](#) *tbl, const char *m)

3.76.1 Detailed Description

Provide a write lock on an [ibis::part](#).

The documentation for this class was generated from the following file:

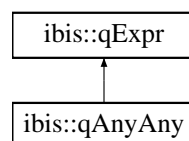
- [part.h](#)

3.77 `ibis::qAnyAny` Class Reference

A user specifies this type of query expression with the following syntax,.

```
#include <qExpr.h>
```

Inheritance diagram for `ibis::qAnyAny`:



Public Member Functions

- virtual `qExpr` * `dup () const`
- const char * `getPrefix () const`

- `const std::vector< double > &getValues () const`
- virtual void `print (std::ostream &out) const`
Print out the node in the string form.
- virtual void `printRange (std::ostream &out) const`
Print out the attribute name and the constants involved in the range expressions.
- `qAnyAny (const char *pre, const char *val)`
Constructing an object of type `qAnyAny` from two strings.

3.77.1 Detailed Description

A user specifies this type of query expression with the following syntax,.

- `any(prefix) = value`
- `any(prefix) in (list of values)` If any column with the given prefix contains the specified values, the row is considered as a hit. This is intended to be used in the cases where the `prefix` is actually the name of a set-valued attribute, such as `triggerID` in STAR datasets. In this case, the set-valued attribute is translated into a number of columns with the same prefix. A common query is "does the set contain a particular value?" or "does the set contain a particular set of values?"

3.77.2 Member Function Documentation

3.77.2.1 `void ibis::qAnyAny::printRange (std::ostream & out) const` [virtual]

Print out the attribute name and the constants involved in the range expressions.

Only print something if the node is a `qRange`.

Reimplemented from `ibis::qExpr`.

The documentation for this class was generated from the following files:

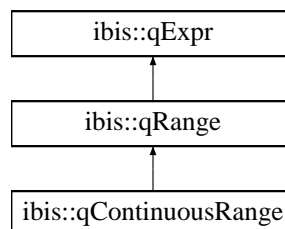
- `qExpr.h`
- `qExpr.cpp`

3.78 `ibis::qContinuousRange` Class Reference

Simple range condition.

```
#include <qExpr.h>
```

Inheritance diagram for `ibis::qContinuousRange`:



Public Member Functions

- virtual const char * `colName` () const
Returns the name of the attribute involved.
- virtual `qContinuousRange` * `dup` () const
- virtual bool `empty` () const
Is the current range empty?
- void `foldBoundaries` ()
- void `foldUnsignedBoundaries` ()
- virtual bool `inRange` (double val) const
Given a value, determine whether it is in the range defined.
- double & `leftBound` ()
- virtual const double & `leftBound` () const
The lower bound of the range.
- `COMPARE` & `leftOperator` ()
- `COMPARE` `leftOperator` () const
- bool `operator<` (const `qContinuousRange` &y) const
The operator< for `ibis::qContinuousRange`.
- virtual void `print` (std::ostream &) const
Print out the node in the string form.
- virtual void `printRange` (std::ostream &out) const
Print out the attribute name and the constants involved in the range expressions.
- `qContinuousRange` (const char *prop, `COMPARE` op, double val)
- `qContinuousRange` (double lv, `COMPARE` lop, const char *prop, `COMPARE` rop, double rv)
- `qContinuousRange` (const `qContinuousRange` &rhs)
- `qContinuousRange` (const char *col, `COMPARE` op, uint32_t val)
- `qContinuousRange` (const char *lstr, `COMPARE` lop, const char *prop, `COMPARE` rop, const char *rstr)
- virtual void `restrictRange` (double left, double right)
Reduce the range to be no more than [left, right].
- double & `rightBound` ()
- virtual const double & `rightBound` () const
The upper bound of the range.
- `COMPARE` & `rightOperator` ()
- `COMPARE` `rightOperator` () const

3.78.1 Detailed Description

Simple range condition.

It is implemented as a derived class of `qExpr`. Possible range operator are defined in `ibis::qExpr::COMPARE`.

3.78.2 Member Function Documentation

3.78.2.1 `bool ibis::qContinuousRange::inRange (double val) const` `[inline, virtual]`

Given a value, determine whether it is in the range defined.

Return true if it is, return false, otherwise.

Implements [ibis::qRange](#).

3.78.2.2 `void ibis::qContinuousRange::printRange (std::ostream & out) const` `[virtual]`

Print out the attribute name and the constants involved in the range expressions.

Only print something if the node is a [qRange](#).

Reimplemented from [ibis::qExpr](#).

The documentation for this class was generated from the following files:

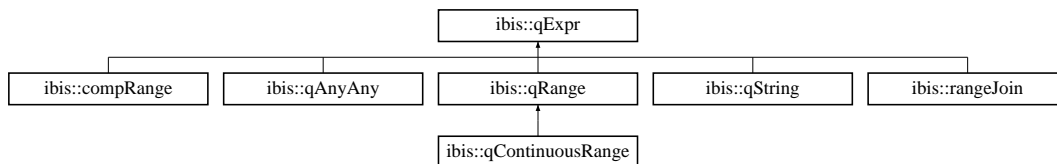
- [qExpr.h](#)
- [qExpr.cpp](#)

3.79 `ibis::qExpr` Class Reference

The top level query expression object.

```
#include <qExpr.h>
```

Inheritance diagram for `ibis::qExpr`:



Public Types

- enum `COMPARE` {
`OP_UNDEFINED`, `OP_LT`, `OP_GT`, `OP_LE`,
`OP_GE`, `OP_EQ` }
Comparison operator supported in RANGE.
- enum `TYPE` {
`LOGICAL_UNDEFINED`, `LOGICAL_NOT`, `LOGICAL_AND`, `LOGICAL_OR`,
`LOGICAL_XOR`, `LOGICAL_MINUS`, `RANGE`, `DRANGE`,
`STRING`, `MSTRING`, `COMPRANGE`, `MATHTERM`,
`JOIN`, `TOPK`, `ANYANY` }
Definition of node types.

Public Member Functions

- virtual `qExpr * dup () const`

- void **extractJoins** (std::vector< const **rangeJoin** * > &terms) const
Extract conjunctive terms of the specified type.
- **qRange** * **findRange** (const char *vname)
Find the first range condition involving the named variable.
- const **qExpr** * **getLeft** () const
Return a const pointer to the left child.
- **qExpr** *& **getLeft** ()
Return a pointer to the left child.
- const **qExpr** * **getRight** () const
Return a const pointer to the right child.
- **qExpr** *& **getRight** ()
Return a pointer to the right child.
- **TYPE** **getType** () const
Return the node type.
- bool **hasJoin** () const
Return true is there is a term with join operation, false otherwise.
- virtual bool **isSimple** () const
Is the expression simple, i.e., containing only simple range conditions joined with logical operators ?
- const **qExpr** & **operator=** (const **qExpr** &rhs)
- virtual void **print** (std::ostream &) const
Print out the node in the string form.
- virtual void **printRange** (std::ostream &out) const
Print out the attribute name and the constants involved in the range expressions.
- **qExpr** (const **qExpr** &qe)
Deep copy.
- **qExpr** (**TYPE** op, **qExpr** *qe1, **qExpr** *qe2)
Construct a full specified node. All three arguments are present.
- **qExpr** (**TYPE** op)
Construct a node of specified type. Not for implicit type conversion.
- **qExpr** ()
Default constructor. It generates a node of undefined type.
- double **reorder** (const **weight** &)
Reorder the expressions tree.
- int **separateSimple** (**ibis::qExpr** *&simple, **ibis::qExpr** *&tail) const
Separate an expression tree into two joined with an AND operator.
- void **setLeft** (**qExpr** *expr)

Change the left child.

- void `setRight (qExpr *expr)`

Change the right child.

- virtual `~qExpr ()`

Destruct the node recursively.

Static Public Member Functions

- static void `simplify (ibis::qExpr *&)`

Attempt to convert simple compRanges into qRanges.

Classes

- struct `weight`

A functor to be used by the function reorder.

3.79.1 Detailed Description

The top level query expression object.

It encodes the logical operations between two child expressions. It is to serve as the interior nodes of an expression tree. Leaf node are going to be derived later.

3.79.2 Member Enumeration Documentation

3.79.2.1 enum `ibis::qExpr::TYPE`

Definition of node types.

Logical operators are listed in the front and leaf node types are listed at the end.

3.79.3 Member Function Documentation

3.79.3.1 virtual void `ibis::qExpr::printRange (std::ostream & out) const` `[inline, virtual]`

Print out the attribute name and the constants involved in the range expressions.

Only print something if the node is a `qRange`.

Reimplemented in `ibis::qContinuousRange`, and `ibis::qAnyAny`.

3.79.3.2 int `ibis::qExpr::separateSimple (ibis::qExpr *& simple, ibis::qExpr *& tail) const`

Separate an expression tree into two joined with an AND operator.

The first one of the two new expressions contains only simple range expressions, and the second contains what is left.

3.79.3.3 `void ibis::qExpr::simplify (ibis::qExpr *&) [static]`

Attempt to convert simple `compRanges` into `qRanges`.

This is necessary because the parser always generates `compRange` instead of `qRange`.

The documentation for this class was generated from the following files:

- [qExpr.h](#)
- [qExpr.cpp](#)

3.80 `ibis::qExpr::weight` Struct Reference

A functor to be used by the function `reorder`.

```
#include <qExpr.h>
```

Public Member Functions

- virtual double `operator()` (const `qExpr *ex`) const=0

3.80.1 Detailed Description

A functor to be used by the function `reorder`.

The documentation for this struct was generated from the following file:

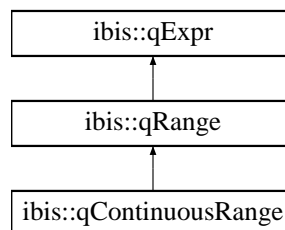
- [qExpr.h](#)

3.81 `ibis::qRange` Class Reference

A class to represent simple range conditions.

```
#include <qExpr.h>
```

Inheritance diagram for `ibis::qRange`:



Public Member Functions

- virtual const char * `colName` () const=0
Returns the name of the attribute involved.
- virtual bool `empty` () const=0
Is the current range empty?
- virtual bool `inRange` (double val) const=0
Given a value, determine whether it is in the range defined.

- virtual const double & `leftBound` () const=0
The lower bound of the range.
- virtual void `restrictRange` (double left, double right)=0
Reduce the range to be no more than [left, right].
- virtual const double & `rightBound` () const=0
The upper bound of the range.

Protected Member Functions

- `qRange` (TYPE t)

3.81.1 Detailed Description

A class to represent simple range conditions.

This is an abstract base class for `qContinuousRange` and `qDiscreteRange`. The two main virtual functions defined in this class are used by the procedures that evaluate the conditions.

3.81.2 Member Function Documentation

3.81.2.1 virtual bool `ibis::qRange::inRange` (double *val*) const [pure virtual]

Given a value, determine whether it is in the range defined.

Return true if it is, return false, otherwise.

Implemented in `ibis::qContinuousRange`.

The documentation for this class was generated from the following file:

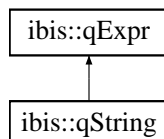
- `qExpr.h`

3.82 `ibis::qString` Class Reference

The class `qString` encapsulates information for comparing string values.

```
#include <qExpr.h>
```

Inheritance diagram for `ibis::qString`:



Public Member Functions

- virtual `qString` * `dup` () const
- const char * `leftString` () const
- virtual void `print` (std::ostream &) const

Print out the node in the string form.

- `qString` (const char *ls, const char *rs)
- const char * `rightString` () const

3.82.1 Detailed Description

The class `qString` encapsulates information for comparing string values.

Only equality comparison is supported at this point. It does not ensure the names are valid in any way. When the check does happen, the left side will be checked first. If it matches the name of a `ibis::column`, the right side will be assumed to be the value one is trying to match. If the left side does not match any known column name, but the right side does, the right side will be assumed to name of column to be searched and the left side will be the value to search against. If neither matches the name of any column, the expression will evaluate to NULL (i.e., no hit).

The documentation for this class was generated from the following files:

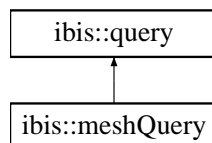
- [qExpr.h](#)
- [qExpr.cpp](#)

3.83 `ibis::query` Class Reference

A data structure for representing user queries.

```
#include <query.h>
```

Inheritance diagram for `ibis::query`:



Public Types

- enum `QUERY_STATE` {
`UNINITIALIZED`, `SET_COMPONENTS`, `SET_RIDS`, `SET_PREDICATE`,
`SPECIFIED`, `QUICK_ESTIMATE`, `FULL_EVALUATE`, `BUNDLES_TRUNCATED`,
`HITS_TRUNCATED` }

Public Member Functions

- void `clear` ()
Releases the resources held by the query object and re-initialize the select clause and the where clause to blank.
- void `clearErrorMessage` () const
Reset the last error message to blank.
- const `selected` & `components` () const
Return a list of names specified in the select clause.
- void `contractQuery` ()
Contracts where clause to preferred bounds.

- long `countHits ()` const
Count the number of hits.
- const `part * dataTable ()` const
Return the pointer to the data table used to process the query.
- const char * `dir ()` const
The query token. For persistent data.
- int `estimate ()`
Functions to perform estimation and retrieve range of hits Computes a lower and an upper bound of hits.
- int `evaluate (const bool evalSelect=false)`
Computes the exact hits.
- void `expandQuery ()`
Expands where clause to preferred bounds.
- const `ibis::bitvector * getHitVector ()` const
Return the pointer to the internal hit vector.
- const char * `getLastError ()` const
Return the last error message recorded internally.
- long `getMaxNumHits ()` const
Return the number of records in the upper bound.
- long `getMinNumHits ()` const
Return the number of records in the lower bound.
- long `getNumHits ()` const
Return the number of records in the exact solution.
- `array_t< double > * getQualifiedDoubles (const char *column_name)`
- `array_t< float > * getQualifiedFloats (const char *column_name)`
- `array_t< int32_t > * getQualifiedInts (const char *column_name)`
The functions getQualifiedXXX return the values of selected columns in the records that satisfies the specified conditions.
- `array_t< uint32_t > * getQualifiedUInts (const char *column_name)`
- `RIDSet * getRIDs (const ibis::bitvector &mask)` const
Return a list of row ids that match the mask.
- `RIDSet * getRIDs ()` const
Return the list of row IDs of the hits.
- const `RIDSet * getRIDsInBundle (const uint32_t bid)` const
Return the list of row IDs of the hits within the specified bundle.
- virtual const char * `getSelectClause ()` const
Return the select clause string.

- `QUERY_STATE` `getState ()` const
Return the current state of query.
- const `RIDSet` * `getUserRIDs ()` const
Return a const pointer to the copy of the user supplied RID set.
- const char * `getWhereClause ()` const
Return the where clause string.
- const char * `id ()` const
Functions about the identity of the query.
- long int `limit` (const char *names, int direction, uint32_t keep, bool updateHits=true)
Truncate the bundles to provide the top-K rows of the bundles.
- void `logMessage` (const char *event, const char *fmt,...) const
Used to print information about the progress or state of query processing.
- int `orderby` (const char *names, int direction) const
Re-order the bundles according the the new "ORDER BY" specification.
- void `printSelected` (std::ostream &out) const
Print the values of the selected columns to the specified output stream.
- void `printSelectedWithRID` (std::ostream &out) const
Print the values of the columns in the select clause without functions.
- `query` (const char *uid=0, const `part` *et=0, const char *pref=0)
Constructor. Generates a new query on the given table et.
- `query` (const char *dir, const `ibis::partList` &tl)
Constructor.
- `RIDSet` * `readRIDs ()` const
- std::string `removeComplexConditions ()`
Separate out the sub-expressions that are not simple.
- `ibis::bitvector` * `sequentialScan ()` const
Return a (new) bitvector that contains the result of directly scan the raw data to determine what records satisfy the user specified conditions.
- int `setRIDs` (const `RIDSet` &set)
Specifies a list of Row IDs for the query object to retrieve the records.
- virtual int `setSelectClause` (const char *str)
Specifies the select clause for the query.
- int `setTable` (const `ibis::part` *tbl)
Resets the table used to evaluate the query conditions to the table specified in the argument.
- int `setWhereClause` (const char *str)
Specifies the where clause for the query.

- `time_t timestamp () const`
The time stamp on the data used to process the query.
- `const char * userName () const`
User started the query.
- `void writeRIDs (const RIDSet *rids) const`
 - `template<> int64_t countDeltaPairs (const array_t< int32_t > &val1, const array_t< uint32_t > &val2, const int32_t &delta) const`
 - `template<> int64_t countDeltaPairs (const array_t< uint32_t > &val1, const array_t< int32_t > &val2, const uint32_t &delta) const`
 - `template<> int64_t countEqualPairs (const array_t< uint32_t > &val1, const array_t< int32_t > &val2) const`
 - `template<> int64_t countEqualPairs (const array_t< int32_t > &val1, const array_t< uint32_t > &val2) const`
This is an explicit specialization of a protected member of `ibis::query` class.
 - `template<> int64_t recordDeltaPairs (const array_t< int32_t > &val1, const array_t< uint32_t > &val2, const array_t< uint32_t > &ind1, const array_t< uint32_t > &ind2, const int32_t &delta, const char *filename) const`
 - `template<> int64_t recordDeltaPairs (const array_t< uint32_t > &val1, const array_t< int32_t > &val2, const array_t< uint32_t > &ind1, const array_t< uint32_t > &ind2, const uint32_t &delta, const char *filename) const`
 - `template<> int64_t recordEqualPairs (const array_t< int32_t > &val1, const array_t< uint32_t > &val2, const array_t< uint32_t > &ind1, const array_t< uint32_t > &ind2, const char *filename) const`
 - `template<> int64_t recordEqualPairs (const array_t< uint32_t > &val1, const array_t< int32_t > &val2, const array_t< uint32_t > &ind1, const array_t< uint32_t > &ind2, const char *filename) const`

Static Public Member Functions

- `static bool isValidToken (const char *tok)`
Is the given string a valid query token.
- `static void keepQueryRecords ()`
Tell the destructor to leave stored information on disk.
- `static void removeQueryRecords ()`
Tell the destructor to remove all stored information about queries.
- `static unsigned tokenLength ()`
Length of the query token.

Protected Member Functions

- `void addJoinConstraints (ibis::qExpr *&exp0) const`
Add constraints derived from domains of the two join columns.
- `void computeHits ()`
- `template<typename T1, typename T2> int64_t countDeltaPairs (const array_t< T1 > &val1, const array_t< T2 > &val2, const T1 &delta) const`
Assume the two input arrays are sorted in ascending order, count the number of elements that are with delta of each other.

- `template<typename T1, typename T2> int64_t countEqualPairs (const array_t< T1 > &val1, const array_t< T2 > &val2) const`

Assume the two input arrays are sorted in ascending order, count the number of elements that match.

- `uint32_t countPages (unsigned wordsize) const`
- `int doContract (ibis::qExpr *exp0) const`
- `void doEstimate (const qExpr *term, ibis::bitvector &low, ibis::bitvector &high) const`
- `void doEvaluate (const qExpr *term, const ibis::bitvector &mask, ibis::bitvector &hits) const`
- `void doEvaluate (const qExpr *term, ibis::bitvector &hits) const`
- `int doExpand (ibis::qExpr *exp0) const`
- `void doScan (const qExpr *term, ibis::bitvector &hits) const`
- `void doScan (const qExpr *term, const ibis::bitvector &mask, ibis::bitvector &hits) const`
- `void gainReadAccess (const char *mesg) const`
- `void gainWriteAccess (const char *mesg) const`
- `void getBounds ()`
- `bool hasBundles () const`
- `void logError (const char *event, const char *fmt,...) const`
- `void logWarning (const char *event, const char *fmt,...) const`
- `int64_t mergePairs (const char *pairfile) const`
- `void orderPairs (const char *pairfile) const`

Sort the content of the file as `ibis::rid_t`.

- `void printRIDs (const RIDSet &ridset) const`

This function prints a list of RIDs to stdout.

- `int64_t processJoin ()`

Process the join operation and return the number of pairs.

- `void readHits ()`
- `void readQuery (const ibis::partList &tl)`
- `template<typename T1, typename T2> int64_t recordDeltaPairs (const array_t< T1 > &val1, const array_t< T2 > &val2, const array_t< uint32_t > &ind1, const array_t< uint32_t > &ind2, const T1 &delta, const char *pairfile) const`
- `template<typename T1, typename T2> int64_t recordEqualPairs (const array_t< T1 > &val1, const array_t< T2 > &val2, const array_t< uint32_t > &ind1, const array_t< uint32_t > &ind2, const char *pairfile) const`

- `void releaseAccess (const char *mesg) const`
- `void removeFiles ()`
- `void reorderExpr ()`
- `int64_t sortEquiJoin (const ibis::rangeJoin &cmp, const ibis::bitvector &mask, const char *pairfile) const`

Perform equi-join by sorting the selected values.

- `int64_t sortEquiJoin (const ibis::rangeJoin &cmp, const ibis::bitvector &mask) const`

Performing an equi-join by sorting the selected values first.

- `int64_t sortJoin (const ibis::rangeJoin &cmp, const ibis::bitvector &mask) const`
- `int64_t sortJoin (const std::vector< const ibis::rangeJoin * > &terms, const ibis::bitvector &mask) const`
- `int64_t sortRangeJoin (const ibis::rangeJoin &cmp, const ibis::bitvector &mask, const char *pairfile) const`

Performing range join by sorting the selected values.

- `int64_t sortRangeJoin (const ibis::rangeJoin &cmp, const ibis::bitvector &mask) const`

Performing a range join by sorting the selected values.

- `int verifyPredicate (qExpr *&qexpr)`
- `void writeHits () const`
- `virtual void writeQuery ()`

Protected Attributes

- `selected comps`
Names of selected components.
- `char * condition`
Query condition (string).
- `ibis::part::readLock * dslock`
A read lock on the table0.
- `ibis::bitvector * hits`
Solution in bitvector form (or lower bound).
- `char lastError [MAX_LINE+PATH_MAX]`
The warning/error message.
- `QUERY_STATE state`
Status of the query.
- `ibis::bitvector * sup`
Estimated upper bound.
- `char * user`
Name of the user who specified the query.

Friends

- class `readLock`
- class `writeLock`

Classes

- class `readLock`
- class `result`
The class `ibis::query::result` allows user to retrieve query result one row at a time.
- class `weight`
- class `writeLock`

3.83.1 Detailed Description

A data structure for representing user queries.

This is the primary entry for user to take advantage of bitmap indexing facilities. A query is a very limited version of the SQL SELECT statement. It is only defined on one table and it takes a where clause and a select clause. The where clause is mandatory. It contains a list of range conditions joined together with logical operators, such as "temperature > 700 and 100 <= pressure < 350". Records whose attribute values satisfy the conditions defined in the where clause is considered hits. A query may retrieve values of variables/columns specified in the select clause. A select clause is optional. If specified, it contains a list of column names. These attributes must not be NULL in order for a record to be a hit. The select clause may also contain column names appearing as the argument to one of the four functions: `avg`, `max`, `min` and `sum`. For example, "temperature, pressure, average(ho2_concentration)" may be a select statement for a Chemistry application.

The hits can be computed in two ways by using functions `estimate` or `evaluate`. The function `estimate` can take advantage of the indices to give two approximate solutions, one as an upper bound and the other as a lower bound. The bitmap indices will be automatically built according to the specification if they are not present. The accuracy of the bounds depend on the nature of the indices available. If no index can be constructed, the lower bound would be empty and the upper bound would include every record. When the function `evaluate` is called, the exact solution is computed no matter whether the function `estimate` has been called or not. The solution produced is recorded as a bit vector. The user may use `ibis::bitvector::indexSet` to extract the record numbers of the hits or use one of the functions `getQualifiedInts`, `getQualifiedFloats`, and `getQualifiedDoubles` to retrieve the values of the selected attributes. Additionally, one may call either `printSelected` or `printSelectedWithRID` to print the selected values to the specified I/O stream.

3.83.2 Constructor & Destructor Documentation

3.83.2.1 `ibis::query::query (const char * dir, const ibis::partList & tl)`

Constructor.

Reconstructs query from stored information in the named directory `dir`. This is only used for recovering from program crashes.

3.83.2.2 `ibis::query::query (const char * uid = 0, const part * et = 0, const char * pref = 0)`

Constructor. Generates a new query on the given table `et`.

If recovery is desired or the query objects has its own special prefix, a cache directory is created to store some information about the query such as the query conditions and the resulting solutions. The stored information enables it to be reconstructed in case of crash.

3.83.3 Member Function Documentation

3.83.3.1 `void ibis::query::contractQuery ()`

Contracts where clause to preferred bounds.

Similar to function `expandQuery`, but makes the bounds of the range conditions narrower rather than wider.

3.83.3.2 `template<typename T1, typename T2> int64_t ibis::query::countDeltaPairs (const array_t< T1 > & val1, const array_t< T2 > & val2, const T1 & delta) const` [protected]

Assume the two input arrays are sorted in ascending order, count the number of elements that are with delta of each other.

Note that both template arguments should be elemental types or they must support operators `-`, `+`, `==` and `<` with mixed types.

3.83.3.3 `int64_t ibis::query::countEqualPairs (const array_t< int32_t > & val1, const array_t< uint32_t > & val2) const`

This is an explicit specialization of a protected member of `ibis::query` class.

Note:

The C++ language rules require explicit specialization of template member function be declared in the namespace containing the function, not inside the class! This apparently causes them to be listed as public functions in Doxygen document.

3.83.3.4 `template<typename T1, typename T2> int64_t ibis::query::countEqualPairs (const array_t< T1 > & val1, const array_t< T2 > & val2) const` `[protected]`

Assume the two input arrays are sorted in ascending order, count the number of elements that match.

Note that both template arguments should be elemental types or they must support operators `==` and `<` with mixed types.

3.83.3.5 `long ibis::query::countHits () const`

Count the number of hits.

Don't generate the hit vector if not already there.

3.83.3.6 `int ibis::query::estimate ()`

Functions to perform estimation and retrieve range of hits Computes a lower and an upper bound of hits.

This is done by using the indices. If possible it will build new indices. The lower bound contains only records that are hits and the upper bound contains all hits but may also contain some records that are not hits. Returns 0 for success, a negative value for error.

3.83.3.7 `int ibis::query::evaluate (const bool evalSelect = false)`

Computes the exact hits.

The same answer shall be computed whether there is any index or not. The argument `evalSelect` indicates whether the select clause should be evaluated at the same time. If its value is true, the columns specified in the select clause will be retrieved from disk and stored in the temporary location for this query. If not, the qualified values will be retrieved from disk when one of `getRIDs`, `getQualifiedInts`, `getQualifiedFloats`, and `getQualifiedDoubles` is issued. In the later case, only the specified column is retrieved. In addition, the values of column at the time of the function are read, which can be potentially different different from the time when the function evaluate was called.

Returns 0 for success, a negative value for error.

See also:

[getQualifiedInts](#)

3.83.3.8 `void ibis::query::expandQuery ()`

Expands where clause to preferred bounds.

This is to make sure the function estimate will give exact answer. It does nothing if there is no preferred bounds in the indices.

3.83.3.9 `const ibis::bitvector* ibis::query::getHitVector () const` `[inline]`

Return the pointer to the internal hit vector.

The user should NOT attempt to free the returned pointer.

3.83.3.10 `array_t< int32_t > * ibis::query::getQualifiedInts (const char * column_name)`

The functions `getQualifiedXXX` return the values of selected columns in the records that satisfies the specified conditions.

The caller must call the operator `delete` to free the pointers returned.

Note:

Any column of the table may be specified, not just those given in the select clause. The content returned is read from disk when these functions are called, which may be different from their values when the function `evaluate` was called. In other word, they may be inconsistent with the conditions specified in the where clause. For append-only data, this is NOT an issue.

The above caveat also applies to the two versions of `getRIDs`.

3.83.3.11 `ibis::RIDSet * ibis::query::getRIDs (const ibis::bitvector & mask) const`

Return a list of row ids that match the mask.

The user is responsible for freeing the pointer.

See also:

[getQualifiedInts](#)

3.83.3.12 `ibis::RIDSet * ibis::query::getRIDs () const`

Return the list of row IDs of the hits.

The user is responsible for freeing the pointer.

See also:

[getQualifiedInts](#)

3.83.3.13 `bool ibis::query::isValidToken (const char * tok) [static]`

Is the given string a valid query token.

Return true if it has the expected token format, otherwise false.

3.83.3.14 `long int ibis::query::limit (const char * names, int direction, uint32_t keep, bool updateHits = true)`

Truncate the bundles to provide the top-K rows of the bundles.

It returns the number of results kept, which is the smaller of the current number of bundles and the input argument `keep`. A negative value is returned in case of error, e.g., query has not been fully specified. If the second argument is true, the internal hit vector is updated to match the truncated solution. Otherwise, the internal hit vector is left unchanged. Since the functions `getNumHits` and `getQualifiedXXX` uses this internal hit vector, it is generally a good idea to update the hit vector. On the other hand, one may wish to avoid this update if the hit vector is not used in any way.

3.83.3.15 `void ibis::query::logMessage (const char * event, const char * fmt, ...) const`

Used to print information about the progress or state of query processing.

It prefixes each message with a query token.

3.83.3.16 `int ibis::query::orderby (const char * names, int direction) const`

Re-order the bundles according the the new "ORDER BY" specification.

It returns 0 if it completes successfully. It returns a negative number to indicate error. If `direction` ≥ 0 , sort the values in ascending order, otherwise, sort them in descending order.

3.83.3.17 `void ibis::query::orderPairs (const char * pfile) const` [protected]

Sort the content of the file as `ibis::rid_t`.

It reads the content of the file one block at a time during the initial sorting of the blocks. It then merges the sorted blocks to produce an overall sorted file. Note that `ibis::rid_t` is simply a pair of integers. Since the pairs are recorded as pairs of integers too, this should work.

3.83.3.18 `void ibis::query::printSelected (std::ostream & out) const`

Print the values of the selected columns to the specified output stream.

The printed values are grouped by the columns without functions. For each group, the functions are evaluated on the columns named in the function. This is equivalent to having implicit "GROUP BY" and "ORDER BY" keywords on all columns appears without a function in the select clause.

3.83.3.19 `void ibis::query::printSelectedWithRID (std::ostream & out) const`

Print the values of the columns in the select clause without functions.

One of the groups of unique values are printed. For each group, the row ID (RID) of the rows are also printed.

3.83.3.20 `int64_t ibis::query::processJoin ()` [protected]

Process the join operation and return the number of pairs.

Additionally, it performs only self-join, i.e., join a table with itself. This is only meant to test some algorithms for evaluating joins.

3.83.3.21 `std::string ibis::query::removeComplexConditions ()`

Separate out the sub-expressions that are not simple.

This is intended to allow the overall where clause to be evaluated in separated steps, where the simple conditions are left for this software to handle and the more complex ones are to be handled by another software. The set of conditions remain with this query object and the conditions returned by this function are assumed to be connected with the operator AND. If the top-most operator in the WHERE clause is not an AND operator, the whole clause will be returned if it contains any conditions that is not simple, otherwise, an empty string will be returned.

3.83.3.22 `ibis::bitvector * ibis::query::sequentialScan () const`

Return a (new) bitvector that contains the result of directly scanning the raw data to determine what records satisfy the user specified conditions.

It is mostly used for testing purposes. It can be called any time after the where clause is set, and does not change the state of the current query.

3.83.3.23 `int ibis::query::setSelectClause (const char * str)` [virtual]

Specifies the select clause for the query.

The select clause is a string of attribute names (plus the four predefined functions, `avg`, `max`, `min` and `sum`) separated by spaces, commas (,) or semicolons(;). Repeated calls to this function simply overwrite the previous definition of the select clause. If no select clause is specified, the where clause alone determines whether a record is a hit or not. The select clause will be reordered to make the plain column names without functions appear before with functions.

3.83.3.24 `int ibis::query::setTable (const ibis::part * tbl)`

Resets the table used to evaluate the query conditions to the table specified in the argument.

-1: nil pointer to table or empty table. -2: invalid string for select clause. -3: select clause contains invalid column name. -4: invalid string for where clause. -5: where clause can not be parsed correctly. -6: where clause contains invalid column names or unsupported functions. -7: empty rid list for set rid operation. -8: neither rids nor range conditions are set. -9: encountered some exceptional conditions during query evaluations. -10: no private directory to store bundles. -11: Query not fully evaluated.

3.83.3.25 `int ibis::query::setWhereClause (const char * str)`

Specifies the where clause for the query.

The where clause is of a string for a list of range conditions. A where clause is mandatory if a query is to be estimated or evaluated. This function may be called multiple times and each invocation will overwrite the previous where clause.

3.83.3.26 `int64_t ibis::query::sortEquiJoin (const ibis::rangeJoin & cmp, const ibis::bitvector & mask, const char * pairfile) const` [protected]

Perform equi-join by sorting the selected values.

This version reads the values marked to be 1 in the bitvector `mask`. It writes the the pairs satisfying the join condition to a file name `pairfile`.

3.83.3.27 `int64_t ibis::query::sortEquiJoin (const ibis::rangeJoin & cmp, const ibis::bitvector & mask) const` [protected]

Performing an equi-join by sorting the selected values first.

This version reads the values marked to be 1 in the bitvector `mask` and performs the actual operation of counting the number of pairs with equal values in memory.

3.83.3.28 `int64_t ibis::query::sortRangeJoin (const ibis::rangeJoin & cmp, const ibis::bitvector & mask) const` [protected]

Performing a range join by sorting the selected values.

The sorting is performed through `std::sort` algorithm.

The documentation for this class was generated from the following files:

- [query.h](#)
- [query.cpp](#)

3.84 `ibis::query::result` Class Reference

The class `ibis::query::result` allows user to retrieve query result one row at a time.

```
#include <bundle.h>
```

Public Member Functions

- `uint32_t colPosition (const char *cname) const`
- `double getDouble (uint32_t selind) const`

Retrieve the value of column `selind` in the select clause as a double-precision floating-point number.

- `double getDouble (const char *cname) const`

Retrieve the value of the named column as a double-precision floating-point number.

- float `getFloat` (uint32_t selind) const
Retrieve the value of column `selind` in the select clause as a single-precision floating-point number.
- float `getFloat` (const char *cname) const
Retrieve the value of the named column as a single-precision floating-point number.
- int `getInt` (uint32_t selind) const
Retrieve the value of column `selind` in the select clause as a signed integer.
- int `getInt` (const char *cname) const
Retrieve the value of the named column as a signed integer.
- std::string `getString` (uint32_t selind) const
Retrieve the string value.
- std::string `getString` (const char *cname) const
- unsigned `getUInt` (uint32_t selind) const
Retrieve the value of column `selind` in the select clause as an unsigned integer.
- unsigned `getUInt` (const char *cname) const
Retrieve the value of the named column as an unsigned integer.
- bool `next` ()
- void `reset` ()
Move the internal pointer back to the beginning.
- **result** (`ibis::query` &q)

3.84.1 Detailed Description

The class `ibis::query::result` allows user to retrieve query result one row at a time.

It matches the semantics of an ODBC cursor. That is the function `next` has to be called before the first set of results can be used.

Note:

This implementation stores the results in memory. Therefore, it is not suitable for handling large result sets.

3.84.2 Member Function Documentation

3.84.2.1 `int ibis::query::result::getInt (uint32_t selind) const` [inline]

Retrieve the value of column `selind` in the select clause as a signed integer.

Note:

Since this version avoids the name look up, it should be more efficient than the version taking the column name as argument.

3.84.2.2 `int ibis::query::result::getInt (const char * cname) const`

Retrieve the value of the named column as a signed integer.

Note:

The name must appeared in the select clause of the query used to construct the `result` object.

3.84.2.3 `std::string ibis::query::result::getString (uint32_t selind) const` [inline]

Retrieve the string value.

See also:

[ibis::bundle::getString](#) for limitations.

3.84.2.4 `void ibis::query::result::reset ()`

Move the internal pointer back to the beginning.

Must call `next` to use the first set of results.

The documentation for this class was generated from the following files:

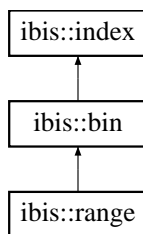
- [bundle.h](#)
- [bundle.cpp](#)

3.85 `ibis::range` Class Reference

The range encoded bitmap index based.

```
#include <ibin.h>
```

Inheritance diagram for `ibis::range`:



Public Member Functions

- long **append** (const [ibis::range](#) &tail)
- virtual long **append** (const char *dt, const char *df, uint32_t nnew)
Extend the index.
- virtual void **binBoundaries** (std::vector< double > &) const
The function binBoundaries and binWeights return bin boundaries and counts of each bin respectively.
- virtual void **binWeights** (std::vector< uint32_t > &) const
- virtual int **contractRange** ([ibis::qContinuousRange](#) &range) const
- virtual uint32_t **estimate** (const [ibis::qContinuousRange](#) &expr) const
Returns an upper bound on the number of hits.
- virtual void **estimate** (const [ibis::qContinuousRange](#) &expr, [ibis::bitvector](#) &lower, [ibis::bitvector](#) &upper) const
Computes an approximation of hits as a pair of lower and upper bounds.
- virtual long **evaluate** (const [ibis::qContinuousRange](#) &expr, [ibis::bitvector](#) &hits) const
To evaluate the exact hits.

- virtual int `expandRange` (`ibis::qContinuousRange &range`) const
The functions `expandRange` and `contractRange` expands or contracts the boundaries of a range condition so that the new range will have exact answers using the function estimate.
- virtual double `getMax` () const
The maximum value recorded in the index.
- virtual double `getSum` () const
Compute the approximate sum of all the values indexed.
- virtual const char * `name` () const
Returns the name of the index, similar to the function `type`, but returns a string instead.
- virtual uint32_t `numBins` () const
- virtual void `print` (`std::ostream &out`) const
Prints human readable information.
- **range** (`const ibis::bin &rhs`)
- **range** (`const ibis::column *c`, `ibis::fileManager::storage *st`, `uint32_t offset=8`)
- **range** (`const ibis::column *c=0`, `const char *f=0`)
- void `read` (`int fdes`, `uint32_t offset`, `const char *fname`)
Read an `ibis::range` embedded with multiple data structures.
- virtual void `read` (`ibis::fileManager::storage *st`)
Reconstructs an index from an array of bytes.
- virtual void `read` (`const char *idxfile`)
Reconstructs an index from the named file.
- virtual void `speedTest` (`std::ostream &out`) const
Time some logical operations and print out their speed.
- virtual INDEX_TYPE `type` () const
Returns an index type identifier.
- virtual float `undecidable` (`const ibis::qContinuousRange &expr`, `ibis::bitvector &iffy`) const
Mark the position of the rows that can not be decided with this index.
- virtual void `write` (`const char *dt`) const
Save index to a file.

Protected Member Functions

- virtual double **computeSum** () const
- virtual void **locate** (`const ibis::qContinuousRange &expr`, `uint32_t &cand0`, `uint32_t &cand1`, `uint32_t &hit0`, `uint32_t &hit1`) const
- virtual void **locate** (`const ibis::qContinuousRange &expr`, `uint32_t &cand0`, `uint32_t &cand1`) const
- virtual uint32_t **locate** (`const double &val`) const

Protected Attributes

- double `max1`
- double `min1`

Friends

- class `ibis::pale`

3.85.1 Detailed Description

The range encoded bitmap index based.

It can be thought of as a cumulative version of `ibis::bin`, where the *i*th bit vector marks the possibles of all entries where $x < \text{bounds}[i]$.

3.85.2 Member Function Documentation

3.85.2.1 `void ibis::range::estimate (const ibis::qContinuousRange & expr, ibis::bitvector & lower, ibis::bitvector & upper) const` [virtual]

Computes an approximation of hits as a pair of lower and upper bounds.

Parameters:

expr the query expression to be evaluated.

lower a bitvector marking a subset of the hits. All rows marked with one (1) are definitely hits.

upper a bitvector marking a superset of the hits. All hits are marked with one, but some of the rows marked one may not be hits. If the variable *upper* is empty, the variable *lower* is assumed to contain the exact answer.

Reimplemented from [ibis::bin](#).

3.85.2.2 `long ibis::range::evaluate (const ibis::qContinuousRange & expr, ibis::bitvector & hits) const` [virtual]

To evaluate the exact hits.

On success, return the number of hits, otherwise a negative value is returned.

Reimplemented from [ibis::bin](#).

3.85.2.3 `int ibis::range::expandRange (ibis::qContinuousRange & range) const` [virtual]

The functions `expandRange` and `contractRange` expands or contracts the boundaries of a range condition so that the new range will have exact answers using the function estimate.

The default implementation provided does nothing since this is only meaningful for indices based on bins.

Reimplemented from [ibis::bin](#).

3.85.2.4 `double ibis::range::getSum () const` [virtual]

Compute the approximate sum of all the values indexed.

If it decides that computing the sum directly from the vertical partition is more efficient, it will return NaN immediately.

Reimplemented from [ibis::bin](#).

3.85.2.5 `void ibis::range::print (std::ostream & out) const` [virtual]

Prints human readable information.

Outputs information about the index as text to the specified output stream.

Reimplemented from [ibis::bin](#).

3.85.2.6 `void ibis::range::read (ibis::fileManager::storage * st)` [virtual]

Reconstructs an index from an array of bytes.

Intended for internal use only!

Reimplemented from [ibis::bin](#).

3.85.2.7 `void ibis::range::read (const char * idxfile)` [virtual]

Reconstructs an index from the named file.

The name can be the directory containing an index file. In this case, the name of the index file must be the name of the column followed by ".idx" suffix.

Reimplemented from [ibis::bin](#).

3.85.2.8 `float ibis::range::undecidable (const ibis::qContinuousRange & expr, ibis::bitvector & iffy) const` [virtual]

Mark the position of the rows that can not be decided with this index.

Parameters:

expr the range conditions to be evaluated.

iffy the bitvector marking the positions of rows that can not be decided using the index. Return value is the expected fraction of undecided rows that might satisfy the range conditions.

Reimplemented from [ibis::bin](#).

3.85.2.9 `void ibis::range::write (const char * dt) const` [virtual]

Save index to a file.

Outputs the index in a compact binary format to the named file or directory. The index file contains a header that can be identified by the function `isIndex`.

Reimplemented from [ibis::bin](#).

The documentation for this class was generated from the following files:

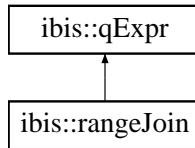
- [ibin.h](#)
- [irange.cpp](#)

3.86 `ibis::rangeJoin` Class Reference

A join is defined by two names and a numerical expression.

```
#include <qExpr.h>
```

Inheritance diagram for `ibis::rangeJoin`:



Public Member Functions

- virtual `rangeJoin * dup ()` const
- const char * `getName1 ()` const
- const char * `getName2 ()` const
- const `ibis::compRange::term * getRange ()` const
- `ibis::compRange::term * getRange ()`
- virtual void `print (std::ostream &out)` const
Print out the node in the string form.
- `rangeJoin (const char *n1, const char *n2, ibis::compRange::term *x)`
- `rangeJoin (const char *n1, const char *n2)`
- void `setRange (ibis::compRange::term *t)`

3.86.1 Detailed Description

A join is defined by two names and a numerical expression.

If the numerical expression is not specified, it is a standard equal-join, 'name1 = name2'. If the numerical expression is specified, it is a range-join, 'name1 between name2 - expr and name2 + expr'.

The documentation for this class was generated from the following files:

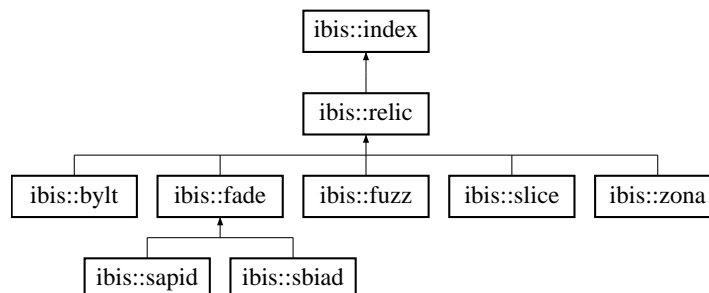
- [qExpr.h](#)
- [qExpr.cpp](#)

3.87 `ibis::relic` Class Reference

The basic bitmap index.

```
#include <irelic.h>
```

Inheritance diagram for `ibis::relic`:



Public Member Functions

- long `append (const array_t< uint32_t > &ind)`

Append a list of integers.

- long **append** (const [ibis::relic](#) &tail)
- virtual long **append** (const char *dt, const char *df, uint32_t nnew)

Extend the index.

- virtual void **binBoundaries** (std::vector< double > &b) const

The function binBoundaries and binWeights return bin boundaries and counts of each bin respectively.

- virtual void **binWeights** (std::vector< uint32_t > &b) const
- virtual int64_t **estimate** (const [ibis::relic](#) &idx2, const [ibis::rangeJoin](#) &expr, const [ibis::bitvector](#) &mask, const [ibis::qRange](#) *const range1, const [ibis::qRange](#) *const range2) const
- virtual int64_t **estimate** (const [ibis::relic](#) &idx2, const [ibis::rangeJoin](#) &expr, const [ibis::bitvector](#) &mask) const

Estimate an upper bound for the number of pairs produced from marked records.

- virtual void **estimate** (const [ibis::relic](#) &idx2, const [ibis::rangeJoin](#) &expr, const [ibis::bitvector](#) &mask, const [ibis::qRange](#) *const range1, const [ibis::qRange](#) *const range2, [ibis::bitvector64](#) &lower, [ibis::bitvector64](#) &upper) const

Note:

It is assumed that range1 is for column 1 in the join expression and range2 is for column 2 in the join expression.

- virtual void **estimate** (const [ibis::relic](#) &idx2, const [ibis::rangeJoin](#) &expr, const [ibis::bitvector](#) &mask, [ibis::bitvector64](#) &lower, [ibis::bitvector64](#) &upper) const

Estimate the pairs for the range join operator.

- virtual uint32_t **estimate** (const [ibis::qDiscreteRange](#) &expr) const
- virtual void **estimate** (const [ibis::qDiscreteRange](#) &expr, [ibis::bitvector](#) &lower, [ibis::bitvector](#) &upper) const

Estimate the hits for discrete ranges, i.e., those translated from 'a IN (x, y, .

- virtual uint32_t **estimate** (const [ibis::qContinuousRange](#) &expr) const

Returns an upper bound on the number of hits.

- virtual void **estimate** (const [ibis::qContinuousRange](#) &expr, [ibis::bitvector](#) &lower, [ibis::bitvector](#) &upper) const

Computes an approximation of hits as a pair of lower and upper bounds.

- virtual double **estimateCost** (const [ibis::qDiscreteRange](#) &expr) const
- virtual double **estimateCost** (const [ibis::qContinuousRange](#) &expr) const

Estimate the code of evaluate a range condition.

- virtual long **evaluate** (const [ibis::qDiscreteRange](#) &expr, [ibis::bitvector](#) &hits) const
- virtual long **evaluate** (const [ibis::qContinuousRange](#) &expr, [ibis::bitvector](#) &hits) const

To evaluate the exact hits.

- virtual long **getCumulativeDistribution** (std::vector< double > &bds, std::vector< uint32_t > &cts) const

Cumulative distribution of the data.

- virtual long **getDistribution** (std::vector< double > &bds, std::vector< uint32_t > &cts) const

Binned distribution of the data.

- virtual double **getMax** () const

The maximum value recorded in the index.

- virtual double `getMin ()` const
The minimum value recorded in the index.
- virtual double `getSum ()` const
Compute the approximate sum of all the values indexed.
- `array_t< uint32_t > * keys` (const `ibis::bitvector` &mask) const
- virtual const char * `name ()` const
Returns the name of the index, similar to the function `type`, but returns a string instead.
- virtual void `print` (std::ostream &out) const
Prints human readable information.
- virtual void `read` (`ibis::fileManager::storage` *st)
Reconstructs an index from an array of bytes.
- virtual void `read` (const char *idxfile)
Reconstructs an index from the named file.
- `relic` (const `ibis::column` *c, `ibis::fileManager::storage` *st, uint32_t offset=8)
The content of the file (following the 8-byte header) is `nrows(uint32_t)` – number of bits in each bit sequences `nobs(uint32_t)` – number of bit sequences `card(uint32_t)` – the number of distinct values, i.e., cardinality (padding to ensure the next data element is on 8-byte boundary) values (`double[card]`) – the values as doubles offset (`uint32_t[nobs+1]`) – the starting positions of the bit sequences (as bit vectors) `bitvectors` – the bitvectors one after another.
- `relic` (const `ibis::column` *c, uint32_t card, `array_t< uint32_t >` &ints)
Construct an index from an integer array.
- `relic` (const `ibis::column` *c, uint32_t popu, uint32_t ntpl=0)
Construct a dummy index.
- `relic` (const `ibis::column` *c, const char *f=0)
- virtual void `speedTest` (std::ostream &out) const
Time some logical operations and print out their speed.
- virtual INDEX_TYPE `type ()` const
Returns an index type identifier.
- virtual float `undecidable` (const `ibis::qDiscreteRange` &expr, `ibis::bitvector` &iffy) const
- virtual float `undecidable` (const `ibis::qContinuousRange` &expr, `ibis::bitvector` &iffy) const
This class and its derived classes should produce exact answers, therefore no undecidable rows.
- virtual void `write` (const char *dt) const
Save index to a file.

Protected Member Functions

- virtual void `clear ()`
Clear the existing content.
- virtual double `computeSum ()` const

- `template<typename E> void construct (const array_t< E > &arr)`
Construct an index from in-memory values.
- `void locate (const ibis::qContinuousRange &expr, uint32_t &hit0, uint32_t &hit1) const`
- `uint32_t locate (const double &val) const`
- `void write (int fdes) const`

Protected Attributes

- `array_t< double > vals`

3.87.1 Detailed Description

The basic bitmap index.

It generates one bitmap for each distinct value.

3.87.2 Constructor & Destructor Documentation

3.87.2.1 `ibis::relic::relic (const ibis::column * c, uint32_t popu, uint32_t ntpl = 0)`

Construct a dummy index.

All entries have the same value `popu`. This is used to generate index for meta tags from STAR data.

3.87.3 Member Function Documentation

3.87.3.1 `long ibis::relic::append (const array_t< uint32_t > &ind)`

Append a list of integers.

The integers are treated as bin numbers. This function is primarily used by `ibis::category::append()`.

3.87.3.2 `template<typename E> void ibis::relic::construct (const array_t< E > &arr)` [protected]

Construct an index from in-memory values.

The type `E` is intended to be element types supported in `column.h`.

3.87.3.3 `void ibis::relic::estimate (const ibis::relic &idx2, const ibis::rangeJoin &expr, const ibis::bitvector &mask, const ibis::qRange *const range1, const ibis::qRange *const range2, ibis::bitvector64 &lower, ibis::bitvector64 &upper) const` [virtual]

Note:

It is assumed that `range1` is for column 1 in the join expression and `range2` is for column 2 in the join expression.

No name matching is performed.

3.87.3.4 `void ibis::relic::estimate (const ibis::relic &idx2, const ibis::rangeJoin &expr, const ibis::bitvector &mask, ibis::bitvector64 &lower, ibis::bitvector64 &upper) const` [virtual]

Estimate the pairs for the range join operator.

Only records that are masked are evaluated.

3.87.3.5 `virtual void ibis::relic::estimate (const ibis::qDiscreteRange & expr, ibis::bitvector & lower, ibis::bitvector & upper) const` [`inline, virtual`]

Estimate the hits for discrete ranges, i.e., those translated from 'a IN (x, y, . . .)'.
 Reimplemented from [ibis::index](#).

3.87.3.6 `virtual void ibis::relic::estimate (const ibis::qContinuousRange & expr, ibis::bitvector & lower, ibis::bitvector & upper) const` [`inline, virtual`]

Computes an approximation of hits as a pair of lower and upper bounds.

Parameters:

expr the query expression to be evaluated.

lower a bitvector marking a subset of the hits. All rows marked with one (1) are definitely hits.

upper a bitvector marking a superset of the hits. All hits are marked with one, but some of the rows marked one may not be hits. If the variable upper is empty, the variable lower is assumed to contain the exact answer.

Implements [ibis::index](#).

Reimplemented in [ibis::slice](#).

3.87.3.7 `long ibis::relic::evaluate (const ibis::qContinuousRange & expr, ibis::bitvector & hits) const` [`virtual`]

To evaluate the exact hits.

On success, return the number of hits, otherwise a negative value is returned.

Implements [ibis::index](#).

Reimplemented in [ibis::slice](#), [ibis::fade](#), [ibis::sbiad](#), [ibis::sapid](#), [ibis::fuzz](#), [ibis::bylt](#), and [ibis::zona](#).

3.87.3.8 `double ibis::relic::getSum () const` [`virtual`]

Compute the approximate sum of all the values indexed.

If it decides that computing the sum directly from the vertical partition is more efficient, it will return NaN immediately.

Implements [ibis::index](#).

Reimplemented in [ibis::slice](#), and [ibis::fade](#).

3.87.3.9 `void ibis::relic::print (std::ostream & out) const` [`virtual`]

Prints human readable information.

Outputs information about the index as text to the specified output stream.

Implements [ibis::index](#).

Reimplemented in [ibis::slice](#), [ibis::fade](#), [ibis::sbiad](#), [ibis::sapid](#), [ibis::fuzz](#), [ibis::bylt](#), and [ibis::zona](#).

3.87.3.10 `void ibis::relic::read (ibis::fileManager::storage * st)` [`virtual`]

Reconstructs an index from an array of bytes.

Intended for internal use only!

Implements [ibis::index](#).

Reimplemented in [ibis::slice](#), [ibis::fade](#), [ibis::fuzz](#), [ibis::bylt](#), and [ibis::zona](#).

3.87.3.11 `void ibis::relic::read (const char * idxfile)` [virtual]

Reconstructs an index from the named file.

The name can be the directory containing an index file. In this case, the name of the index file must be the name of the column followed by ".idx" suffix.

Implements `ibis::index`.

Reimplemented in `ibis::slice`, `ibis::fade`, `ibis::fuzz`, `ibis::bylt`, and `ibis::zona`.

3.87.3.12 `void ibis::relic::write (const char * dt) const` [virtual]

Save index to a file.

Outputs the index in a compact binary format to the named file or directory. The index file contains a header that can be identified by the function `isIndex`.

Implements `ibis::index`.

Reimplemented in `ibis::slice`, `ibis::fade`, `ibis::sbiad`, `ibis::sapid`, `ibis::fuzz`, `ibis::bylt`, and `ibis::zona`.

The documentation for this class was generated from the following files:

- `irelic.h`
- `irelic.cpp`

3.88 `ibis::resource` Class Reference

A container for name-value pairs.

```
#include <resource.h>
```

Public Types

- `typedef std::map< const char *, resource *, ibis::lessi > gList`

The name-value pairs are categorized into two types, names that map to simple values (vList) and names that map to groups of name-value pairs (gList).

- `typedef std::map< const char *, char *, ibis::lessi > vList`

Public Member Functions

- `void add (const char *name, const char *value)`

Insert a new name-value pair.

- `bool empty () const`

Returns true if there is no name-value pair on record.

- `gList::const_iterator gBegin () const`
- `gList::const_iterator gEnd () const`
- `const resource * getGroup (const char *name) const`

Find a group with the given name.

- `double getNumber (const char *name) const`

Parse the string value as a number.

- `std::string getPrefix () const`

Return the name of the full prefix of the resource.

- `const char * getValue (const char *name) const`
Find a named parameter.
- `bool isTrue (const char *name) const`
If the named parameter exists and its value is one of "true", "yes", "on" or "1", this function will return true, otherwise false.
- `const resource & operator= (const resource &rhs)`
The assignment operator.
- `const char * operator\[\] (const char *name) const`
This group of functions search through multiple levels of the name hierarchy.
- `void read (const char *fn=0)`
Read the content of the file and add it to the existing lists of name-value pairs.
- `resource (const resource &rhs)`
- `resource (const resource *ctx, const char *pfx)`
- `resource (const char *fn=0)`
- `vList::const_iterator vBegin () const`
- `vList::const_iterator vEnd () const`
- `void write (const char *fn=0) const`
Write the name-value pairs to the named file.

Static Public Member Functions

- `static void clear (vList &vl)`
Clear a vList.
- `static bool isStringTrue (const char *val)`
Returns true is the string value should be interpreted as logical truth.
- `static void parseNameValuePairs (const char *in, vList &lst)`
Parse a string into a name-value list.

3.88.1 Detailed Description

A container for name-value pairs.

It is mostly used for storing the configuration parameters. The parameters are in a format as follows: group:group:...:name=value where the delimiter can be either '*', ':' or '.' and anything following the first '=' sign is assumed to be part of the value string until the end of line. The leading and trailing spaces are removed from both the name and the value. The specification that appears later in the same configuration file or read later (through a call to [read\(\)](#)) will overwrite the parameter with the same name. This include the groups and individual parameters. For example if a parameter named 'abcd' is specified first, but later, the same name is used as a group name, then the previously specified parameter will be removed from the list and the new group will be inserted. If the parameter with name 'abcd' appeared again, then the group 'abcd' will be removed and the named parameter will be inserted.

The line length must of no more than MAX_LINE defined in [const.h](#).

The top level group name can be any one of the following: all, common, and '*'. When writing out the parameters, the top level name is not written.

3.88.2 Member Function Documentation

3.88.2.1 `void ibis::resource::add (const char * name, const char * value)`

Insert a new name-value pair.

It replaces the existing value.

3.88.2.2 `const ibis::resource * ibis::resource::getGroup (const char * name) const` [inline]

Find a group with the given name.

The name is expected to be a simple name without any separators. Any separator in the name will cause it to return a nil pointer.

3.88.2.3 `double ibis::resource::getNumber (const char * name) const`

Parse the string value as a number.

If the first non-numeric character is a 'k' or 'm' or 'g', the proceeding number is multiplied by 1024, 1048576, or 1073742824. If the first non-numeric character is 'h', the value before it is multiplied by 3600 (h for hour), converting it from hours to seconds.

3.88.2.4 `const char * ibis::resource::getValue (const char * name) const` [inline]

Find a named parameter.

The name is expected to be a simple name without any separators. Any separator in it will cause a nil pointer to be returned.

3.88.2.5 `const char * ibis::resource::operator[] (const char * name) const`

This group of functions search through multiple levels of the name hierarchy.

The `operator[]` returns a pointer to the string value, `getNumber` returns the string value as a number, and `isTrue` returns the string value as boolean.

The incoming name can contain multiple separators. Each component of the name is separated by one separator. From the left to right, the left-most component defines the highest level of the hierarchy. A high-level name forms the context for the next level of the name hierarchy. The final component of the name is directly associated with a string value. The search algorithm first descend to the lowest level with the matching names and starts to look for a name that matches the last component of the specified name. If a match is not found, it will go back one level and perform the same search. This continues until a match is found or it has searched all the levels.

3.88.2.6 `void ibis::resource::parseNameValuePairs (const char * in, vList & lst) [static]`

Parse a string into a name-value list.

Add the new ones to the existing list.

3.88.2.7 `void ibis::resource::read (const char * fn = 0)`

Read the content of the file and add it to the existing lists of name-value pairs.

It will read the first file in the following list and add the content to the existing list of parameter,

- (1) argument to this function (fn),
- (2) environment variable IBISRC,
- (3) file named `ibis.rc` in the current directory,

- (4) file named `.ibisrc` in the current directory,
- (5) file named `.ibisrc` in the user's home directory.

It will attempt to parse the content of the first file it finds. The content of the file is parsed and added to the current content of the resource object. The parameters with the same will overwrite the existing values. If it can not find any one of the files, it will return without modifying the current content of the resource object.

3.88.2.8 `void ibis::resource::write (const char *fn = 0) const`

Write the name-value pairs to the named file.

If the file name is a nil pointer, the pairs are written to the standard output. If it can not open the named file, it will also write to the standard output.

The documentation for this class was generated from the following files:

- [resource.h](#)
- `resource.cpp`

3.89 `ibis::rid_t` Union Reference

The object identifiers used to distinguish records.

```
#include <const.h>
```

Public Member Functions

- `bool operator!= (const rid_t &r) const`
- `bool operator< (const rid_t &r) const`
- `bool operator<= (const rid_t &r) const`
- `bool operator== (const rid_t &r) const`
- `bool operator> (const rid_t &r) const`
- `bool operator>= (const rid_t &r) const`

Public Attributes

- `ibis::rid_t::name num`
As two 32-bit values.
- `uint64_t value`
As a single 64-bit value.

Classes

- `struct name`
As two 32-bit values.

3.89.1 Detailed Description

The object identifiers used to distinguish records.

The documentation for this union was generated from the following file:

- [const.h](#)

3.90 `ibis::rid_t::name` Struct Reference

As two 32-bit values.

```
#include <const.h>
```

Public Attributes

- `uint32_t event`
Event number. Less significant.
- `uint32_t run`
Run number. More significant.

3.90.1 Detailed Description

As two 32-bit values.

The documentation for this struct was generated from the following file:

- [const.h](#)

3.91 `ibis::ridHandler` Class Reference

A class for handling file IO for `ibis::rid_t`.

```
#include <rids.h>
```

Public Member Functions

- `int append` (const `RIDSet` &rids, const char *destination) const
Append the rid set to the name file.
- `int read` (`RIDSet` &rids, const char *source)
This function is capable of reading a file written with one write command and multiple append commands.
- `ridHandler` (const char *dbName, const char *pref="ibis")
- `int write` (const `RIDSet` &rids, const char *destination, const char *dbName=0)
Write the rid set.

Protected Member Functions

- `int matchDBName` (std::istream &_from) const
- `int readDBName` (std::istream &_from)
- `int readRidCount` (std::istream &_from, int &ic) const
- `int readVersion` (std::istream &_from) const

Protected Attributes

- char * `_dbName`
- char * `_prefix`
- pthread_mutex_t `mutex`

Static Protected Attributes

- static const char *const [version](#)
Implements the functions defined in `ibis::ridHandler`.

3.91.1 Detailed Description

A class for handling file IO for `ibis::rid_t`.

3.91.2 Member Function Documentation

3.91.2.1 `int ibis::ridHandler::append (const RIDSet & rids, const char * fname) const`

Append the rid set to the name file.

Return the number of rids written. This function can be called after `ibis::ridHandler::write` has been called to write a file. It can be called many times. The function `ibis::ridHandler::read` will concatenate all rid sets into one.

3.91.2.2 `int ibis::ridHandler::read (ibis::RIDSet & rids, const char * fname)`

This function is capable of reading a file written with one write command and multiple append commands.

All rids are placed in

- rids in the order they appear in the file. The member variable
- `_dbName` will be set to the name stored in the file.

3.91.2.3 `int ibis::ridHandler::write (const RIDSet & rids, const char * fname, const char * dbName = 0)`

Write the rid set.

Return the number of rids written. If the first argument is specified, the internally stored `dbName` would be modified.

The documentation for this class was generated from the following files:

- [rids.h](#)
- [rids.cpp](#)

3.92 `ibis::roster` Class Reference

A roster list is a list of indices for ordering the values in the ascending order.

```
#include <iroster.h>
```

Public Member Functions

- const [array_t](#)< uint32_t > & [array](#) () const
- const [ibis::column](#) * [getColumn](#) () const
- template<typename T> void [icSearch](#) (const std::vector< T > &vals, std::vector< uint32_t > &pos) const
- int [locate](#) (const std::vector< double > &vals, [ibis::bitvector](#) &positions) const
- int [locate](#) (const std::vector< float > &vals, [ibis::bitvector](#) &positions) const
- int [locate](#) (const std::vector< uint32_t > &vals, [ibis::bitvector](#) &positions) const
- int [locate](#) (const std::vector< int32_t > &vals, [ibis::bitvector](#) &positions) const

Locate the the values and set their positions in the bitvector.

- const char * **name** () const
- template<typename T> void **oocSearch** (const std::vector< T > &vals, std::vector< uint32_t > &pos) const
- uint32_t **operator[]** (uint32_t i) const
- void **print** (std::ostream &out) const
Output minimal information about the roster list.
- void **read** (ibis::fileManager::storage *st)
- void **read** (const char *idxfile)
- **roster** (const ibis::column *c, const ibis::bitvector &mask, const char *f=0)
Select the values that are marked 1 in mask and sort them.
- **roster** (const ibis::column *c, ibis::fileManager::storage *st, uint32_t offset=8)
For reconstructing the data structure from raw bytes stored in st.
- **roster** (const ibis::column *c, const char *f=0)
Sort all data elements in the named file (default to the one in the current working directory).
- uint32_t **size** () const
- void **write** (const char *dt) const
Write two files, .ind for indices and .srt to the sorted values.
- void **writeSorted** (const char *dt) const
Write the sorted version of the attribute values to a .srt file.

Static Public Member Functions

- template<class T> static long **mergeBlock2** (const char *dsrc, const char *dout, const uint32_t segment, array_t< T > &buf1, array_t< T > &buf2, array_t< T > &buf3)
A two-way merge algorithm.

3.92.1 Detailed Description

A roster list is a list of indices for ordering the values in the ascending order.

It can use external sort if the data and indices can not fit into memory. The indices will be written to a file with .ind extension and if the external sorting procedure is used a file with .srt extension is also generated to store a sorted version of the values. If the indices can not be loaded into memory as a whole, the .ind file will be opened for future operations.

3.92.2 Constructor & Destructor Documentation

3.92.2.1 **ibis::roster::roster** (const **ibis::column** * c, **ibis::fileManager::storage** * st, **uint32_t offset** = 8)

For reconstructing the data structure from raw bytes stored in st.

The content of the file (following the 8-byte header) is the index array ind.

3.92.3 Member Function Documentation

3.92.3.1 `int ibis::roster::locate (const std::vector< int32_t > & vals, ibis::bitvector & positions) const`

Locate the the values and set their positions in the bitvector.

Return a negative value for error, zero or a positive value for in case of success. The input values are assumed to be sorted in ascending order.

3.92.3.2 `template<class T> long ibis::roster::mergeBlock2 (const char * dsrc, const char * dout, const uint32_t segment, array_t< T > & buf1, array_t< T > & buf2, array_t< T > & buf3) [static]`

A two-way merge algorithm.

Uses `std::less<T>` for comparisons. Assumes the sorted segment size is `segment` elements of type `T`.

3.92.3.3 `void ibis::roster::writeSorted (const char * df) const`

Write the sorted version of the attribute values to a `.srt` file.

Attempt to read the whole column into memory first. If it fails to do so, it will read one value at a time from the original data file.

The documentation for this class was generated from the following files:

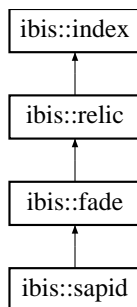
- [iroster.h](#)
- `iroster.cpp`

3.93 `ibis::sapid` Class Reference

The multicomponent equality encoded index.

```
#include <irelic.h>
```

Inheritance diagram for `ibis::sapid`:



Public Member Functions

- virtual long [append](#) (const char *dt, const char *df, uint32_t nnew)
Extend the index.
- virtual long [evaluate](#) (const `ibis::qDiscreteRange` &expr, [ibis::bitvector](#) &hits) const
- virtual long [evaluate](#) (const `ibis::qContinuousRange` &expr, [ibis::bitvector](#) &hits) const
To evaluate the exact hits.
- virtual const char * [name](#) () const

Returns the name of the index, similar to the function `type`, but returns a string instead.

- virtual void `print` (std::ostream &out) const
Prints human readable information.
- `sapid` (const `ibis::column` *c, `ibis::fileManager::storage` *st, uint32_t offset=8)
- `sapid` (const `ibis::column` *c=0, const char *f=0, const uint32_t nbase=2)
- virtual void `speedTest` (std::ostream &out) const
Time some logical operations and print out their speed.
- virtual INDEX_TYPE `type` () const
Returns an index type identifier.
- virtual void `write` (const char *dt) const
Save index to a file.

3.93.1 Detailed Description

The multicomponent equality encoded index.

3.93.2 Member Function Documentation

3.93.2.1 long `ibis::sapid::evaluate` (const `ibis::qContinuousRange` & *expr*, `ibis::bitvector` & *hits*) const
[virtual]

To evaluate the exact hits.

On success, return the number of hits, otherwise a negative value is returned.

Reimplemented from `ibis::fade`.

3.93.2.2 void `ibis::sapid::print` (std::ostream & *out*) const [virtual]

Prints human readable information.

Outputs information about the index as text to the specified output stream.

Reimplemented from `ibis::fade`.

3.93.2.3 void `ibis::sapid::write` (const char * *dt*) const [virtual]

Save index to a file.

Outputs the index in a compact binary format to the named file or directory. The index file contains a header that can be identified by the function `isIndex`.

Reimplemented from `ibis::fade`.

The documentation for this class was generated from the following files:

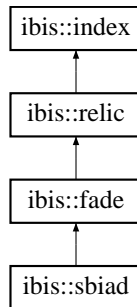
- `irelic.h`
- `isapid.cpp`

3.94 **ibis::sbiad Class Reference**

The multicomponent interval encoded index.

```
#include <irelic.h>
```

Inheritance diagram for `ibis::sbiad`:



Public Member Functions

- virtual long `append` (const char *dt, const char *df, uint32_t nnew)
Extend the index.
- virtual long `evaluate` (const `ibis::qDiscreteRange` &expr, `ibis::bitvector` &hits) const
- virtual long `evaluate` (const `ibis::qContinuousRange` &expr, `ibis::bitvector` &hits) const
To evaluate the exact hits.
- virtual const char * `name` () const
Returns the name of the index, similar to the function `type`, but returns a string instead.
- virtual void `print` (std::ostream &out) const
Prints human readable information.
- `sbiad` (const `ibis::column` *c, `ibis::fileManager::storage` *st, uint32_t offset=8)
- `sbiad` (const `ibis::column` *c=0, const char *f=0, const uint32_t nbase=2)
- virtual void `speedTest` (std::ostream &out) const
Time some logical operations and print out their speed.
- virtual INDEX_TYPE `type` () const
Returns an index type identifier.
- virtual void `write` (const char *dt) const
Save index to a file.

3.94.1 Detailed Description

The multicomponent interval encoded index.

Defined by Chan and Ioannidis (SIGMOD 99).

3.94.2 Member Function Documentation

3.94.2.1 long `ibis::sbiad::evaluate` (const `ibis::qContinuousRange` & expr, `ibis::bitvector` & hits) const
[virtual]

To evaluate the exact hits.

On success, return the number of hits, otherwise a negative value is returned.

Reimplemented from [ibis::fade](#).

3.94.2.2 `void ibis::sbiad::print (std::ostream & out) const` [virtual]

Prints human readable information.

Outputs information about the index as text to the specified output stream.

Reimplemented from [ibis::fade](#).

3.94.2.3 `void ibis::sbiad::write (const char * dt) const` [virtual]

Save index to a file.

Outputs the index in a compact binary format to the named file or directory. The index file contains a header that can be identified by the function `isIndex`.

Reimplemented from [ibis::fade](#).

The documentation for this class was generated from the following files:

- [irelic.h](#)
- [isbiad.cpp](#)

3.95 `ibis::selected` Class Reference

A data structure to store the select clause of a query.

```
#include <util.h>
```

Public Types

- typedef `std::vector< std::string >::const_iterator` `const_iterator`
An iterator through the column names of the select clause.
- enum `FUNCTION` {
 `NIL`, `AVG`, `MAX`, `MIN`,
 `SUM` }

Public Member Functions

- `const_iterator` `begin` () const
- void `clear` ()
- bool `empty` () const
- `const_iterator` `end` () const
- `size_t` `find` (const char *key) const
Return the first occurrence of the string.
- `FUNCTION` `getFunction` (size_t i) const
- const char * `getName` (size_t i) const
Access the ith column name of the select clause.
- const `std::vector< std::string >` & `getNames` () const
Return the list of names stored internally.

- `std::string getTerm (size_t i) const`
*Return the *i*th term, with the function name.*
- `size_t nPlain () const`
- `const char * operator * () const`
Output a stringized version of the select clause.
- `const char * operator[] (size_t i) const`
- `void remove (const std::vector< size_t > &ents)`
Remove the entries specified.
- `void select (const std::vector< const char * > &nl, bool sort=true)`
- `void select (const char *str, bool sort=true)`
Parse the select clause. By default, place the functions last.
- `selected (const char *str)`
The default constructor leaves data members empty.
- `size_t size () const`

3.95.1 Detailed Description

A data structure to store the select clause of a query.

A select clause may contain a list of names plus a list of simple functions. The supported functions are `avg`, `max`, `min` and `sum`. Each of these functions can only take a single name as it argument.

3.95.2 Member Function Documentation

3.95.2.1 `size_t ibis::selected::find (const char * key) const`

Return the first occurrence of the string.

Returns the value of `size` if the given `key` is not in the list of selected components.

3.95.2.2 `const char* ibis::selected::getName (size_t i) const` `[inline]`

Access the *i*th column name of the select clause.

In case of a function, it returns the name of the argument rather than the whole function. This operator is only intended to be used to extract the column values.

The documentation for this class was generated from the following files:

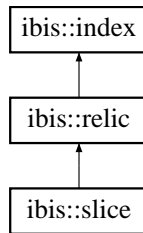
- [util.h](#)
- [query.cpp](#)

3.96 `ibis::slice` Class Reference

The bit-sliced index (O'Neil). It used the binary encoding.

```
#include <irelic.h>
```

Inheritance diagram for `ibis::slice`:



Public Member Functions

- virtual long `append` (const char *dt, const char *df, uint32_t nnew)
Extend the index.
- virtual void `binWeights` (std::vector< uint32_t > &b) const
- virtual uint32_t `estimate` (const `ibis::qContinuousRange` &expr) const
Returns an upper bound on the number of hits.
- virtual void `estimate` (const `ibis::qContinuousRange` &expr, `ibis::bitvector` &lower, `ibis::bitvector` &upper) const
Computes an approximation of hits as a pair of lower and upper bounds.
- virtual double `estimateCost` (const `ibis::qDiscreteRange` &expr) const
- virtual double `estimateCost` (const `ibis::qContinuousRange` &expr) const
Estimate the code of evaluate a range condition.
- virtual long `evaluate` (const `ibis::qDiscreteRange` &expr, `ibis::bitvector` &hits) const
- virtual long `evaluate` (const `ibis::qContinuousRange` &expr, `ibis::bitvector` &hits) const
To evaluate the exact hits.
- virtual double `getSum` () const
Compute the approximate sum of all the values indexed.
- virtual const char * `name` () const
Returns the name of the index, similar to the function `type`, but returns a string instead.
- virtual void `print` (std::ostream &out) const
Prints human readable information.
- virtual void `read` (`ibis::fileManager::storage` *st)
Reconstruct an index from a piece of consecutive memory.
- virtual void `read` (const char *idxfile)
Read the index contained in the file f.
- `slice` (const `ibis::column` *c, `ibis::fileManager::storage` *st, uint32_t offset=8)
- `slice` (const `ibis::column` *c=0, const char *f=0)
- virtual void `speedTest` (std::ostream &out) const
Time some logical operations and print out their speed.
- virtual INDEX_TYPE `type` () const
Returns an index type identifier.

- virtual void `write` (const char *dt) const
Save index to a file.

Protected Member Functions

- virtual void `clear` ()
Clear the existing content.

3.96.1 Detailed Description

The bit-sliced index (O'Neil). It used the binary encoding.

3.96.2 Member Function Documentation

3.96.2.1 void `ibis::slice::estimate` (const `ibis::qContinuousRange` & *expr*, `ibis::bitvector` & *lower*, `ibis::bitvector` & *upper*) const [virtual]

Computes an approximation of hits as a pair of lower and upper bounds.

Parameters:

expr the query expression to be evaluated.

lower a bitvector marking a subset of the hits. All rows marked with one (1) are definitely hits.

upper a bitvector marking a superset of the hits. All hits are marked with one, but some of the rows marked one may not be hits. If the variable upper is empty, the variable lower is assumed to contain the exact answer.

Reimplemented from `ibis::relic`.

3.96.2.2 long `ibis::slice::evaluate` (const `ibis::qContinuousRange` & *expr*, `ibis::bitvector` & *hits*) const [virtual]

To evaluate the exact hits.

On success, return the number of hits, otherwise a negative value is returned.

Reimplemented from `ibis::relic`.

3.96.2.3 double `ibis::slice::getSum` () const [virtual]

Compute the approximate sum of all the values indexed.

If it decides that computing the sum directly from the vertical partition is more efficient, it will return NaN immediately.

Reimplemented from `ibis::relic`.

3.96.2.4 void `ibis::slice::print` (std::ostream & *out*) const [virtual]

Prints human readable information.

Outputs information about the index as text to the specified output stream.

Reimplemented from `ibis::relic`.

3.96.2.5 `void ibis::slice::read (ibis::fileManager::storage * st)` [virtual]

Reconstruct an index from a piece of consecutive memory.

Unlike the implementations for other type indices, this function always reads all bit vectors.

Reimplemented from [ibis::relic](#).

3.96.2.6 `void ibis::slice::read (const char * f)` [virtual]

Read the index contained in the file `f`.

This function always reads all bitvectors.

Reimplemented from [ibis::relic](#).

3.96.2.7 `void ibis::slice::write (const char * dt) const` [virtual]

Save index to a file.

Outputs the index in a compact binary format to the named file or directory. The index file contains a header that can be identified by the function `isIndex`.

Reimplemented from [ibis::relic](#).

The documentation for this class was generated from the following files:

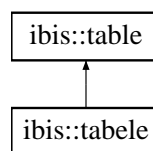
- [irelic.h](#)
- [islice.cpp](#)

3.97 `ibis::tabelle` Class Reference

A trivial class for table with one row and one column.

```
#include <tab.h>
```

Inheritance diagram for `ibis::tabelle`:

**Public Member Functions**

- virtual int [buildIndex](#) (const char *, const char *)
Create the index for named column.
- virtual int [buildIndexes](#) (const char *)
Create indexes for every column of the table.
- const char * [colName](#) () const
- virtual [stringList](#) [columnNames](#) () const
Return column names.
- virtual [typeList](#) [columnTypes](#) () const
Return data types.

- virtual `ibis::table::cursor * createCursor () const`
Create a `CURSOR` object to perform row-wise data access.
- virtual void `describe (std::ostream &) const`
Print a description of the table to the specified output stream.
- virtual int `dump (std::ostream &, const char *) const`
Dump the values in ASCII form to the specified output stream.
- virtual void `estimate (const char *cond, uint64_t &nmin, uint64_t &nmax) const`
Estimate the number of rows satisfying the selection conditions.
- virtual `int64_t getColumnAsBytes (const char *, char *) const`
Retrieve all values of the named column.
- virtual `int64_t getColumnAsDoubles (const char *, double *) const`
Retrieve all values of the named column.
- virtual `int64_t getColumnAsFloats (const char *, float *) const`
Retrieve all values of the named column.
- virtual `int64_t getColumnAsInts (const char *, int32_t *) const`
Retrieve all values of the named column.
- virtual `int64_t getColumnAsLongs (const char *cn, int64_t *vals) const`
Retrieve all values of the named column.
- virtual `int64_t getColumnAsShorts (const char *, int16_t *) const`
Retrieve all values of the named column.
- virtual `int64_t getColumnAsStrings (const char *, std::vector< std::string > &) const`
Retrieve the null-terminated strings as a vector of `std::string` objects.
- virtual `int64_t getColumnAsUBytes (const char *, unsigned char *) const`
Retrieve all values of the named column.
- virtual `int64_t getColumnAsUInts (const char *cn, uint32_t *vals) const`
Retrieve all values of the named column.
- virtual `int64_t getColumnAsULongs (const char *cn, uint64_t *vals) const`
Retrieve all values of the named column.
- virtual `int64_t getColumnAsUShorts (const char *, uint16_t *) const`
Retrieve all values of the named column.
- virtual long `getHistogram (const char *, const char *, double, double, double, std::vector< size_t > &) const`
Compute the histogram of the named column.
- virtual long `getHistogram2D (const char *, const char *, double, double, double, const char *, double, double, double, std::vector< size_t > &) const`
Compute a two-dimension histogram on columns `cname1` and `name2`.

- virtual long [getHistogram3D](#) (const char *, const char *, double, double, double, const char *, double, double, double, const char *, double, double, double, std::vector< size_t > &) const
Compute a three-dimensional histogram on the named columns.
- virtual [table](#) * [groupby](#) (const [stringList](#) &) const
Perform aggregate functions on the current table.
- virtual void [indexSpec](#) (const char *, const char *)
Replace the current indexing option.
- virtual const char * [indexSpec](#) (const char *) const
Retrieve the current indexing option.
- virtual size_t [nColumns](#) () const
- virtual uint64_t [nRows](#) () const
- virtual void [orderby](#) (const [stringList](#) &)
Reorder the rows.
- virtual void [reverseRows](#) ()
Reverse the order of the rows.
- virtual [table](#) * [select](#) (const char *, const char *) const
Given a set of column names and a set of selection conditions, compute another table that represents the selected values.
- **tabele** (uint64_t nr=0, const char *nm=0)
- **tabele** (const char *na, const char *de, uint64_t nr, const char *nm=0)

Friends

- class **cursor**

Classes

- class **cursor**

3.97.1 Detailed Description

A trivial class for table with one row and one column.

This type of table is generated when the select clause is "count(*)". This class could be replaced with [ibis::tabula](#), however, treating the output of "count(*)" as a one-row-and-one-column table is closer to the ODBC/JDBC convention.

3.97.2 Member Function Documentation

3.97.2.1 virtual int [ibis::tabele::buildIndex](#) (const char *, const char *) [inline, virtual]

Create the index for named column.

The existing index will be replaced. If an indexing option is not specified, it will use the internally recorded option for the named column or the table containing the column.

Note:

Unless any there is a specific instruction to not index a column, the querying functions will automatically build indices as necessary. However, as building an index is relatively expensive process, building an index on a column

is on average about four or five times as expensive as reading the column from disk, this function is provided so that it is possible to build indexes beforehand.

Implements [ibis::table](#).

3.97.2.2 `virtual int ibis::table::buildIndexes (const char *) [inline, virtual]`

Create indexes for every column of the table.

Existing indexes will be replaced. If an indexing option is not specified, the internally recorded options will be used. [buildIndex](#)

Implements [ibis::table](#).

3.97.2.3 `int ibis::table::dump (std::ostream &, const char *) const [inline, virtual]`

Dump the values in ASCII form to the specified output stream.

The default delimiter is coma (","), which produces Comma-Separated-Values (CSV).

Implements [ibis::table](#).

3.97.2.4 `void ibis::table::estimate (const char * cond, uint64_t & nmin, uint64_t & nmax) const [inline, virtual]`

Estimate the number of rows satisfying the selection conditions.

The number of rows is between [nmin, nmax].

Implements [ibis::table](#).

3.97.2.5 `virtual int64_t ibis::table::getColumnAsBytes (const char *, char *) const [inline, virtual]`

Retrieve all values of the named column.

The member functions of this class only support access to whole column at a time. Use `table::cursor` class for row-wise accesses. For fixed-width data types, the raw pointers are used to point to the values to be returned. In these cases, the caller is responsible for allocating enough storage for the values to be returned.

Implements [ibis::table](#).

3.97.2.6 `virtual int64_t ibis::table::getColumnAsDoubles (const char *, double *) const [inline, virtual]`

Retrieve all values of the named column.

The member functions of this class only support access to whole column at a time. Use `table::cursor` class for row-wise accesses. For fixed-width data types, the raw pointers are used to point to the values to be returned. In these cases, the caller is responsible for allocating enough storage for the values to be returned.

Implements [ibis::table](#).

3.97.2.7 `virtual int64_t ibis::table::getColumnAsFloats (const char *, float *) const [inline, virtual]`

Retrieve all values of the named column.

The member functions of this class only support access to whole column at a time. Use `table::cursor` class for row-wise accesses. For fixed-width data types, the raw pointers are used to point to the values to be returned. In these cases, the caller is responsible for allocating enough storage for the values to be returned.

Implements [ibis::table](#).

3.97.2.8 `virtual int64_t ibis::table::getColumnAsInts (const char *, int32_t *) const` [inline, virtual]

Retrieve all values of the named column.

The member functions of this class only support access to whole column at a time. Use `table::cursor` class for row-wise accesses. For fixed-width data types, the raw pointers are used to point to the values to be returned. In these cases, the caller is responsible for allocating enough storage for the values to be returned.

Implements `ibis::table`.

3.97.2.9 `virtual int64_t ibis::table::getColumnAsLongs (const char * cn, int64_t * vals) const` [inline, virtual]

Retrieve all values of the named column.

The member functions of this class only support access to whole column at a time. Use `table::cursor` class for row-wise accesses. For fixed-width data types, the raw pointers are used to point to the values to be returned. In these cases, the caller is responsible for allocating enough storage for the values to be returned.

Implements `ibis::table`.

3.97.2.10 `virtual int64_t ibis::table::getColumnAsShorts (const char *, int16_t *) const` [inline, virtual]

Retrieve all values of the named column.

The member functions of this class only support access to whole column at a time. Use `table::cursor` class for row-wise accesses. For fixed-width data types, the raw pointers are used to point to the values to be returned. In these cases, the caller is responsible for allocating enough storage for the values to be returned.

Implements `ibis::table`.

3.97.2.11 `virtual int64_t ibis::table::getColumnAsStrings (const char *, std::vector< std::string > &) const` [inline, virtual]

Retrieve the null-terminated strings as a vector of `std::string` objects.

Both `ibis::CATEGORY` and `ibis::TEXT` types can be retrieved using this function.

Implements `ibis::table`.

3.97.2.12 `virtual int64_t ibis::table::getColumnAsUBytes (const char *, unsigned char *) const` [inline, virtual]

Retrieve all values of the named column.

The member functions of this class only support access to whole column at a time. Use `table::cursor` class for row-wise accesses. For fixed-width data types, the raw pointers are used to point to the values to be returned. In these cases, the caller is responsible for allocating enough storage for the values to be returned.

Implements `ibis::table`.

3.97.2.13 `virtual int64_t ibis::table::getColumnAsUInts (const char * cn, uint32_t * vals) const` [inline, virtual]

Retrieve all values of the named column.

The member functions of this class only support access to whole column at a time. Use `table::cursor` class for row-wise accesses. For fixed-width data types, the raw pointers are used to point to the values to be returned. In these cases, the caller is responsible for allocating enough storage for the values to be returned.

Implements `ibis::table`.

3.97.2.14 `virtual int64_t ibis::table::getColumnAsULongs (const char * cn, uint64_t * vals) const` `[inline, virtual]`

Retrieve all values of the named column.

The member functions of this class only support access to whole column at a time. Use `table::cursor` class for row-wise accesses. For fixed-width data types, the raw pointers are used to point to the values to be returned. In these cases, the caller is responsible for allocating enough storage for the values to be returned.

Implements `ibis::table`.

3.97.2.15 `virtual int64_t ibis::table::getColumnAsUShorts (const char *, uint16_t *) const` `[inline, virtual]`

Retrieve all values of the named column.

The member functions of this class only support access to whole column at a time. Use `table::cursor` class for row-wise accesses. For fixed-width data types, the raw pointers are used to point to the values to be returned. In these cases, the caller is responsible for allocating enough storage for the values to be returned.

Implements `ibis::table`.

3.97.2.16 `virtual long ibis::table::getHistogram (const char *, const char *, double, double, double, std::vector< size_t > &) const` `[inline, virtual]`

Compute the histogram of the named column.

This version uses the user specified bins: `[begin, begin+stride)` `[begin+stride, begin+2*stride)`, A record is placed in bin

$(x - \text{begin}) / \text{stride}$, where the first bin is bin 0. This gives a total of

$(\text{end} - \text{begin}) / \text{stride}$ bins.

Note:

Records (rows) outside of the range `[begin, end]` are not counted.

Non-positive `stride` is considered as an error.

If `end` is less than `begin`, an empty array `counts` is returned along with return value 0.

Implements `ibis::table`.

3.97.2.17 `virtual long ibis::table::getHistogram2D (const char *, const char *, double, double, double, const char *, double, double, double, std::vector< size_t > &) const` `[inline, virtual]`

Compute a two-dimension histogram on columns `cname1` and `name2`.

The bins along each dimension are defined the same way as in function `getHistogram`. The array `counts` stores the two-dimensional bins with the first dimension as the slow varying dimension following C convention for ordering multi-dimensional arrays.

Implements `ibis::table`.

3.97.2.18 `virtual long ibis::table::getHistogram3D (const char *, const char *, double, double, double, const char *, double, double, double, const char *, double, double, double, std::vector< size_t > &) const` `[inline, virtual]`

Compute a three-dimensional histogram on the named columns.

The triplets `<begin, end, stride>` are used the same ways in `getHistogram` and `getHistogram2D`. The three dimensional bins are linearized in `counts` with the first being the slowest varying dimension and the third being the fastest varying dimension following the C convention for ordering multi-dimensional arrays.

Implements `ibis::table`.

3.97.2.19 `virtual table* ibis::tabela::groupby (const stringList &) const` [inline, virtual]

Perform aggregate functions on the current table.

It produces a new table. The list of strings passed to this function are interpreted as a set of names followed by a set of functions. Currently, only functions COUNT, AVG, MIN, MAX, and SUM are supported.

Implements [ibis::table](#).

3.97.2.20 `virtual void ibis::tabela::indexSpec (const char *, const char *)` [inline, virtual]

Replace the current indexing option.

If no column name is specified, it resets the indexing option for the table.

Implements [ibis::table](#).

3.97.2.21 `virtual const char* ibis::tabela::indexSpec (const char *) const` [inline, virtual]

Retrieve the current indexing option.

If no column name is specified, it retrieve the indexing option for the table.

Implements [ibis::table](#).

3.97.2.22 `virtual void ibis::tabela::orderby (const stringList &)` [inline, virtual]

Reorder the rows.

Sort the rows in ascending order of the columns specified in the list of column names. This function is not designated `const` because though it does not change the content in SQL logic, but it may change internal representations.

Note:

If an empty list is passed to this function, it will reorder rows using all columns with the column having the smallest number of distinct values first.

Implements [ibis::table](#).

The documentation for this class was generated from the following file:

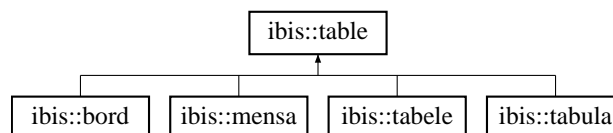
- [tab.h](#)

3.98 `ibis::table` Class Reference

The abstract table class.

```
#include <table.h>
```

Inheritance diagram for `ibis::table`:



Public Types

- `typedef std::map< const char *, ibis::TYPE_T, ibis::lessi > namesTypes`

An associate array of names and types.

- typedef std::vector< const char * > **stringList**
A list of strings.
- typedef std::vector< **ibis::TYPE_T** > **typeList**
A list of data types.

Public Member Functions

- virtual **stringList** **columnNames** () const=0
Return column names.
- virtual **typeList** **columnTypes** () const=0
Return data types.
- virtual **cursor** * **createCursor** () const=0
Create a `CURSOR` object to perform row-wise data access.
- virtual void **describe** (std::ostream &) const=0
Print a description of the table to the specified output stream.
- virtual const char * **description** () const
Free text description.
- virtual int **dump** (std::ostream &out, const char *del=", ") const=0
Dump the values in ASCII form to the specified output stream.
- virtual void **estimate** (const char *cond, uint64_t &nmin, uint64_t &nmax) const =0
Estimate the number of rows satisfying the selection conditions.
- virtual **table** * **groupby** (const char *) const
Perform group-by operation.
- virtual **table** * **groupby** (const **stringList** &) const=0
Perform aggregate functions on the current table.
- virtual const char * **name** () const
Name of the table object.
- virtual size_t **nColumns** () const=0
- virtual uint64_t **nRows** () const=0
- virtual void **orderby** (const char *)
Reorder the rows. The column names are separated by comma.
- virtual void **orderby** (const **stringList** &)=0
Reorder the rows.
- virtual void **reverseRows** ()=0
Reverse the order of the rows.
- virtual **table** * **select** (const char *sel, const char *cond) const=0

Given a set of column names and a set of selection conditions, compute another table that represents the selected values.

- virtual `~table ()`

Destructor.

- virtual int `buildIndex (const char *colname, const char *option=0)=0`
Create the index for named column.
- virtual int `buildIndexes (const char *options=0)=0`
Create indexes for every column of the table.
- virtual void `indexSpec (const char *opt, const char *colname=0)=0`
Replace the current indexing option.
- virtual const char * `indexSpec (const char *colname=0) const=0`
Retrieve the current indexing option.
- virtual int64_t `getColumnAsBytes (const char *cname, char *vals) const=0`
Retrieve all values of the named column.
- virtual int64_t `getColumnAsDoubles (const char *cname, double *vals) const=0`
Retrieve all values of the named column.
- virtual int64_t `getColumnAsFloats (const char *cname, float *vals) const=0`
Retrieve all values of the named column.
- virtual int64_t `getColumnAsInts (const char *cname, int32_t *vals) const=0`
Retrieve all values of the named column.
- virtual int64_t `getColumnAsLongs (const char *cname, int64_t *vals) const=0`
Retrieve all values of the named column.
- virtual int64_t `getColumnAsShorts (const char *cname, int16_t *vals) const=0`
Retrieve all values of the named column.
- virtual int64_t `getColumnAsStrings (const char *cname, std::vector< std::string > &vals) const=0`
Retrieve the null-terminated strings as a vector of std::string objects.
- virtual int64_t `getColumnAsUBytes (const char *cname, unsigned char *vals) const=0`
Retrieve all values of the named column.
- virtual int64_t `getColumnAsUInts (const char *cname, uint32_t *vals) const=0`
Retrieve all values of the named column.
- virtual int64_t `getColumnAsULongs (const char *cname, uint64_t *vals) const=0`
Retrieve all values of the named column.
- virtual int64_t `getColumnAsUShorts (const char *cname, uint16_t *vals) const=0`
Retrieve all values of the named column.
- virtual long `getHistogram (const char *constraints, const char *cname, double begin, double end, double stride, std::vector< size_t > &counts) const =0`
Compute the histogram of the named column.

- virtual long `getHistogram2D` (const char *constraints, const char *cname1, double begin1, double end1, double stride1, const char *cname2, double begin2, double end2, double stride2, std::vector< size_t > &counts) const =0
Compute a two-dimension histogram on columns `cname1` and `name2`.
- virtual long `getHistogram3D` (const char *constraints, const char *cname1, double begin1, double end1, double stride1, const char *cname2, double begin2, double end2, double stride2, const char *cname3, double begin3, double end3, double stride3, std::vector< size_t > &counts) const=0
Compute a three-dimensional histogram on the named columns.

Static Public Member Functions

- static `ibis::table * create` (const char *dir1, const char *dir2)
Create a table object from a pair of data directories.
- static `ibis::table * create` (const char *dir)
Create a table object from the specified data directory.

Protected Member Functions

- void `parseNames` (char *in, `stringList` &out) const
Parse a string into a set of names.
- `table` (const char *na, const char *de)
Copy constructor.
- `table` ()
The default constructor.

Protected Attributes

- std::string `desc_`
Description of the table.
- std::string `name_`
Name of the table.

Classes

- class `cursor`
Cursor class for row-wise data accesses.
- struct `row`
A simple struct for storing a row of a table.

3.98.1 Detailed Description

The abstract table class.

This is an abstract base class that defines the common operations on a data table. Conceptually, data records in a table is organized into rows and columns. A query on a table produces a filtered version of the table. In many database systems this is known as a view on a table. All data tables and views are logically treated as specialization of this `ibis::table` class.

3.98.2 Member Function Documentation

3.98.2.1 `virtual int ibis::table::buildIndex (const char * colname, const char * option = 0) [pure virtual]`

Create the index for named column.

The existing index will be replaced. If an indexing option is not specified, it will use the internally recorded option for the named column or the table containing the column.

Note:

Unless any there is a specific instruction to not index a column, the querying functions will automatically build indices as necessary. However, as building an index is relatively expensive process, building an index on a column is on average about four or five times as expensive as reading the column from disk, this function is provided so that it is possible to build indexes beforehand.

Implemented in `ibis::bord`, `ibis::mensa`, `ibis::tabula`, and `ibis::tabelle`.

3.98.2.2 `virtual int ibis::table::buildIndexes (const char * options = 0) [pure virtual]`

Create indexes for every column of the table.

Existing indexes will be replaced. If an indexing option is not specified, the internally recorded options will be used. `buildIndex`

Implemented in `ibis::bord`, `ibis::mensa`, `ibis::tabula`, and `ibis::tabelle`.

3.98.2.3 `ibis::table * ibis::table::create (const char * dir1, const char * dir2) [static]`

Create a table object from a pair of data directories.

This table maintains two sets of data files so that it can continue to process queries using existing data records while accepting new data records.

3.98.2.4 `virtual int ibis::table::dump (std::ostream & out, const char * del = ", ") const [pure virtual]`

Dump the values in ASCII form to the specified output stream.

The default delimiter is coma (","), which produces Comma-Separated-Values (CSV).

Implemented in `ibis::bord`, `ibis::mensa`, `ibis::tabula`, and `ibis::tabelle`.

3.98.2.5 `virtual void ibis::table::estimate (const char * cond, uint64_t & nmin, uint64_t & nmax) const [pure virtual]`

Estimate the number of rows satisfying the selection conditions.

The number of rows is between `[nmin, nmax]`.

Implemented in `ibis::bord`, `ibis::mensa`, `ibis::tabula`, and `ibis::tabelle`.

3.98.2.6 `virtual int64_t ibis::table::getColumnAsBytes (const char * cname, char * vals) const` [pure virtual]

Retrieve all values of the named column.

The member functions of this class only support access to whole column at a time. Use `table::cursor` class for row-wise accesses. For fixed-width data types, the raw pointers are used to point to the values to be returned. In these cases, the caller is responsible for allocating enough storage for the values to be returned.

Implemented in `ibis::bord`, `ibis::mensa`, `ibis::tabula`, and `ibis::tabelle`.

3.98.2.7 `virtual int64_t ibis::table::getColumnAsDoubles (const char * cname, double * vals) const` [pure virtual]

Retrieve all values of the named column.

The member functions of this class only support access to whole column at a time. Use `table::cursor` class for row-wise accesses. For fixed-width data types, the raw pointers are used to point to the values to be returned. In these cases, the caller is responsible for allocating enough storage for the values to be returned.

Implemented in `ibis::bord`, `ibis::mensa`, `ibis::tabula`, and `ibis::tabelle`.

3.98.2.8 `virtual int64_t ibis::table::getColumnAsFloats (const char * cname, float * vals) const` [pure virtual]

Retrieve all values of the named column.

The member functions of this class only support access to whole column at a time. Use `table::cursor` class for row-wise accesses. For fixed-width data types, the raw pointers are used to point to the values to be returned. In these cases, the caller is responsible for allocating enough storage for the values to be returned.

Implemented in `ibis::bord`, `ibis::mensa`, `ibis::tabula`, and `ibis::tabelle`.

3.98.2.9 `virtual int64_t ibis::table::getColumnAsInts (const char * cname, int32_t * vals) const` [pure virtual]

Retrieve all values of the named column.

The member functions of this class only support access to whole column at a time. Use `table::cursor` class for row-wise accesses. For fixed-width data types, the raw pointers are used to point to the values to be returned. In these cases, the caller is responsible for allocating enough storage for the values to be returned.

Implemented in `ibis::bord`, `ibis::mensa`, `ibis::tabula`, and `ibis::tabelle`.

3.98.2.10 `virtual int64_t ibis::table::getColumnAsLongs (const char * cname, int64_t * vals) const` [pure virtual]

Retrieve all values of the named column.

The member functions of this class only support access to whole column at a time. Use `table::cursor` class for row-wise accesses. For fixed-width data types, the raw pointers are used to point to the values to be returned. In these cases, the caller is responsible for allocating enough storage for the values to be returned.

Implemented in `ibis::bord`, `ibis::mensa`, `ibis::tabula`, and `ibis::tabelle`.

3.98.2.11 `virtual int64_t ibis::table::getColumnAsShorts (const char * cname, int16_t * vals) const` [pure virtual]

Retrieve all values of the named column.

The member functions of this class only support access to whole column at a time. Use `table::cursor` class for row-wise accesses. For fixed-width data types, the raw pointers are used to point to the values to be returned. In these cases, the caller is responsible for allocating enough storage for the values to be returned.

Implemented in `ibis::bord`, `ibis::mensa`, `ibis::tabula`, and `ibis::tabele`.

3.98.2.12 `virtual int64_t ibis::table::getColumnAsStrings (const char * cname, std::vector< std::string > & vals) const` [pure virtual]

Retrieve the null-terminated strings as a vector of `std::string` objects.

Both `ibis::CATEGORY` and `ibis::TEXT` types can be retrieved using this function.

Implemented in `ibis::bord`, `ibis::mensa`, `ibis::tabula`, and `ibis::tabele`.

3.98.2.13 `virtual int64_t ibis::table::getColumnAsUBytes (const char * cname, unsigned char * vals) const` [pure virtual]

Retrieve all values of the named column.

The member functions of this class only support access to whole column at a time. Use `table::cursor` class for row-wise accesses. For fixed-width data types, the raw pointers are used to point to the values to be returned. In these cases, the caller is responsible for allocating enough storage for the values to be returned.

Implemented in `ibis::bord`, `ibis::mensa`, `ibis::tabula`, and `ibis::tabele`.

3.98.2.14 `virtual int64_t ibis::table::getColumnAsUInts (const char * cname, uint32_t * vals) const` [pure virtual]

Retrieve all values of the named column.

The member functions of this class only support access to whole column at a time. Use `table::cursor` class for row-wise accesses. For fixed-width data types, the raw pointers are used to point to the values to be returned. In these cases, the caller is responsible for allocating enough storage for the values to be returned.

Implemented in `ibis::bord`, `ibis::mensa`, `ibis::tabula`, and `ibis::tabele`.

3.98.2.15 `virtual int64_t ibis::table::getColumnAsULongs (const char * cname, uint64_t * vals) const` [pure virtual]

Retrieve all values of the named column.

The member functions of this class only support access to whole column at a time. Use `table::cursor` class for row-wise accesses. For fixed-width data types, the raw pointers are used to point to the values to be returned. In these cases, the caller is responsible for allocating enough storage for the values to be returned.

Implemented in `ibis::bord`, `ibis::mensa`, `ibis::tabula`, and `ibis::tabele`.

3.98.2.16 `virtual int64_t ibis::table::getColumnAsUShorts (const char * cname, uint16_t * vals) const` [pure virtual]

Retrieve all values of the named column.

The member functions of this class only support access to whole column at a time. Use `table::cursor` class for row-wise accesses. For fixed-width data types, the raw pointers are used to point to the values to be returned. In these cases, the caller is responsible for allocating enough storage for the values to be returned.

Implemented in `ibis::bord`, `ibis::mensa`, `ibis::tabula`, and `ibis::tabele`.

3.98.2.17 `virtual long ibis::table::getHistogram (const char * constraints, const char * cname, double begin, double end, double stride, std::vector< size_t > & counts) const` [pure virtual]

Compute the histogram of the named column.

This version uses the user specified bins: [begin, begin+stride) [begin+stride, begin+2*stride), A record is placed in bin

$(x - \text{begin}) / \text{stride}$, where the first bin is bin 0. This gives a total of $(\text{end} - \text{begin}) / \text{stride}$ bins.

Note:

Records (rows) outside of the range `[begin, end]` are not counted.
 Non-positive `stride` is considered as an error.
 If `end` is less than `begin`, an empty array `counts` is returned along with return value 0.

Implemented in `ibis::bord`, `ibis::mensa`, `ibis::tabula`, and `ibis::tabele`.

3.98.2.18 `virtual long ibis::table::getHistogram2D (const char * constraints, const char * cname1, double begin1, double end1, double stride1, const char * cname2, double begin2, double end2, double stride2, std::vector< size_t > & counts) const` `[pure virtual]`

Compute a two-dimension histogram on columns `cname1` and `name2`.

The bins along each dimension are defined the same way as in function `getHistogram`. The array `counts` stores the two-dimensional bins with the first dimension as the slow varying dimension following C convention for ordering multi-dimensional arrays.

Implemented in `ibis::bord`, `ibis::mensa`, `ibis::tabula`, and `ibis::tabele`.

3.98.2.19 `virtual long ibis::table::getHistogram3D (const char * constraints, const char * cname1, double begin1, double end1, double stride1, const char * cname2, double begin2, double end2, double stride2, const char * cname3, double begin3, double end3, double stride3, std::vector< size_t > & counts) const` `[pure virtual]`

Compute a three-dimensional histogram on the named columns.

The triplets `<begin, end, stride>` are used the same ways in `getHistogram` and `getHistogram2D`. The three dimensional bins are linearized in `counts` with the first being the slowest varying dimension and the third being the fastest varying dimension following the C convention for ordering multi-dimensional arrays.

Implemented in `ibis::bord`, `ibis::mensa`, `ibis::tabula`, and `ibis::tabele`.

3.98.2.20 `ibis::table * ibis::table::groupby (const char *) const` `[inline, virtual]`

Perform group-by operation.

The column names and operations are separated by comma.

3.98.2.21 `virtual table* ibis::table::groupby (const stringList &) const` `[pure virtual]`

Perform aggregate functions on the current table.

It produces a new table. The list of strings passed to this function are interpreted as a set of names followed by a set of functions. Currently, only functions COUNT, AVG, MIN, MAX, and SUM are supported.

Implemented in `ibis::bord`, `ibis::mensa`, `ibis::tabula`, and `ibis::tabele`.

3.98.2.22 `virtual void ibis::table::indexSpec (const char * opt, const char * colname = 0)` `[pure virtual]`

Replace the current indexing option.

If no column name is specified, it resets the indexing option for the table.

Implemented in `ibis::bord`, `ibis::mensa`, `ibis::tabula`, and `ibis::tabele`.

3.98.2.23 `virtual const char* ibis::table::indexSpec (const char * colname = 0) const` [pure virtual]

Retrieve the current indexing option.

If no column name is specified, it retrieve the indexing option for the table.

Implemented in `ibis::bord`, `ibis::mensa`, `ibis::tabula`, and `ibis::tabele`.

3.98.2.24 `virtual void ibis::table::orderby (const stringList &) [pure virtual]`

Reorder the rows.

Sort the rows in ascending order of the columns specified in the list of column names. This function is not designated `const` because though it does not change the content in SQL logic, but it may change internal representations.

Note:

If an empty list is passed to this function, it will reorder rows using all columns with the column having the smallest number of distinct values first.

Implemented in `ibis::bord`, `ibis::mensa`, `ibis::tabula`, and `ibis::tabele`.

3.98.2.25 `void ibis::table::parseNames (char * in, stringList & out) const` [protected]

Parse a string into a set of names.

Some bytes may be turned into 0 to mark the end of names or functions.

The documentation for this class was generated from the following files:

- [table.h](#)
- `mensa.cpp`

3.99 `ibis::table::cursor` Class Reference

Cursor class for row-wise data accesses.

```
#include <table.h>
```

Public Member Functions

- virtual `ibis::table::stringList columnNames () const=0`
- virtual `ibis::table::typeList columnTypes () const=0`
- virtual `int dump (std::ostream &out, const char *del=", ") const=0`
Print out the values of the current row.
- virtual `int fetch (uint64_t rownum, ibis::table::row &)=0`
Fetch the content of the specified row and make that row the current row as well.
- virtual `int fetch (ibis::table::row &)=0`
Fetch the content of the next row and make the next row as the current row as well.
- virtual `int fetch (uint64_t rownum)=0`
Make the specified row in the data set available for retrieval.
- virtual `int fetch ()=0`
Make the next row of the data set available for retrieval.

- virtual int `getColumnAsByte` (size_t cnum, char *vals) const=0
This version of `getColumnAsTTT` directly use the column number, i.e., the position of a column in the list returned by function `columnNames` or `columnTypes`.
- virtual int `getColumnAsByte` (const char *cname, char *) const=0
Retrieve the value of the named column.
- virtual int `getColumnAsDouble` (size_t cnum, double *vals) const=0
- virtual int `getColumnAsDouble` (const char *cname, double *) const=0
- virtual int `getColumnAsFloat` (size_t cnum, float *vals) const=0
- virtual int `getColumnAsFloat` (const char *cname, float *) const=0
- virtual int `getColumnAsInt` (size_t cnum, int32_t *vals) const=0
- virtual int `getColumnAsInt` (const char *cname, int32_t *) const =0
- virtual int `getColumnAsLong` (size_t cnum, int64_t *vals) const=0
- virtual int `getColumnAsLong` (const char *cname, int64_t *) const =0
- virtual int `getColumnAsShort` (size_t cnum, int16_t *vals) const=0
- virtual int `getColumnAsShort` (const char *cname, int16_t *) const =0
- virtual int `getColumnAsString` (size_t cnum, std::string &vals) const=0
- virtual int `getColumnAsString` (const char *cname, std::string &) const=0
- virtual int `getColumnAsUByte` (size_t cnum, unsigned char *vals) const=0
- virtual int `getColumnAsUByte` (const char *cname, unsigned char *) const=0
- virtual int `getColumnAsUInt` (size_t cnum, uint32_t *vals) const=0
- virtual int `getColumnAsUInt` (const char *cname, uint32_t *) const=0
- virtual int `getColumnAsULong` (size_t cnum, uint64_t *vals) const=0
- virtual int `getColumnAsULong` (const char *cname, uint64_t *) const=0
- virtual int `getColumnAsUShort` (size_t cnum, uint16_t *vals) const=0
- virtual int `getColumnAsUShort` (const char *cname, uint16_t *) const=0
- virtual uint64_t `getCurrentRowNumber` () const=0
Return the current row number.
- virtual size_t `nColumns` () const=0
- virtual uint64_t `nRows` () const=0

Protected Member Functions

- `cursor` (const `cursor` &)
- `cursor` & `operator=` (const `cursor` &)

3.99.1 Detailed Description

Cursor class for row-wise data accesses.

Note:

Note that this cursor is associated with a table object and can only iterate overall rows of a table. To iterate an arbitrary selection of rows, use the selection to create a new table and then iterate over the new table.

3.99.2 Member Function Documentation

3.99.2.1 virtual int `ibis::table::cursor::fetch` (uint64_t rownum) [pure virtual]

Make the specified row in the data set available for retrieval.

Returns 0 if the specified row is found, returns a negative number to indicate error, such as rownum out of range (-1).

3.99.2.2 `virtual int ibis::table::cursor::fetch ()` [pure virtual]

Make the next row of the data set available for retrieval.

Returns 0 if successful, returns a negative number to indicate error.

3.99.2.3 `virtual int ibis::table::cursor::getColumnAsByte (size_t cnum, char * vals) const` [pure virtual]

This version of `getColumnAsTTT` directly use the column number, i.e., the position of a column in the list returned by function `columnNames` or `columnTypes`.

This version of the data access function may be able to avoid the name lookup and reduce the execution time.

3.99.2.4 `virtual int ibis::table::cursor::getColumnAsByte (const char * cname, char *) const` [pure virtual]

Retrieve the value of the named column.

Note:

Note the cost of name lookup is likely to dominate the total cost of such a function.

3.99.2.5 `virtual uint64_t ibis::table::cursor::getCurrentRowNumber () const` [pure virtual]

Return the current row number.

Rows in a data set are numbered [0

- `nRows () - 1`]. If the cursor is not ready, such as before the first call to `fetch` or function `fetch` returned an error, this function return the same value as function `nRows`.

The documentation for this class was generated from the following file:

- [table.h](#)

3.100 `ibis::table::row` Struct Reference

A simple struct for storing a row of a table.

```
#include <table.h>
```

Public Member Functions

- void `clear ()`
- void `clearValues ()`

Public Attributes

- `std::vector< std::string >` `bytesnames`
For `ibis::BYTE`.
- `std::vector< signed char >` `bytesvalues`
- `std::vector< std::string >` `catsnames`
For `ibis::CATEGORY`.

- `std::vector< std::string > catsvalues`
- `std::vector< std::string > doublesnames`
For `ibis::DOUBLE`.
- `std::vector< double > doublesvalues`
- `std::vector< std::string > floatsnames`
For `ibis::FLOAT`.
- `std::vector< float > floatsvalues`
- `std::vector< std::string > intsnames`
For `ibis::INT`.
- `std::vector< int32_t > intsvvalues`
- `std::vector< std::string > longsnames`
For `ibis::LONG`.
- `std::vector< int64_t > longsvvalues`
- `std::vector< std::string > shortsnames`
For `ibis::SHORT`.
- `std::vector< int16_t > shortsvvalues`
- `std::vector< std::string > textsnames`
For `ibis::TEXT`.
- `std::vector< std::string > textsvvalues`
- `std::vector< std::string > ubytesnames`
For `ibis::UBYTE`.
- `std::vector< unsigned char > ubytesvalues`
- `std::vector< std::string > uintsnames`
For `ibis::UINT`.
- `std::vector< uint32_t > uintsvvalues`
- `std::vector< std::string > ulongsnames`
For `ibis::ULONG`.
- `std::vector< uint64_t > ulongsvvalues`
- `std::vector< std::string > ushortsnames`
For `ibis::USHORT`.
- `std::vector< uint16_t > ushortsvvalues`

3.100.1 Detailed Description

A simple struct for storing a row of a table.

The documentation for this struct was generated from the following file:

- [table.h](#)

3.101 `ibis::tableList` Class Reference

A list of tables.

```
#include <table.h>
```

Public Types

- typedef `tableSet::const_iterator` **iterator**
- typedef `std::map< const char *, ibis::table *, ibis::lessi >` **tableSet**

Public Member Functions

- void **add** (`ibis::table *&tb`)
Add a new table object to the list.
- iterator **begin** () const
Return the iterator to the first table.
- bool **empty** () const
Is the list empty? Returns true if the list is empty, otherwise returns false.
- iterator **end** () const
Return the iterator to the end of the list.
- const `ibis::table * operator[] (const char *tname)` const
Find the named table.
- void **remove** (const char *tname)
Remove the named data table from the list.
- `size_t` **size** () const
Return the number of tables in the list.
- `tableList ()`
Default constructor.
- `~tableList ()`
Destructor. Delete all table objects.

3.101.1 Detailed Description

A list of tables.

It supports simple lookup through `operator[]` and manages the table objects passed to it. Most functions are simply wrappers on `std::map`.

3.101.2 Member Function Documentation

3.101.2.1 `void ibis::tableList::add (ibis::table *&tb)` [inline]

Add a new table object to the list.

Transfers the control of the object to the `tableList`. If the name of the table already exists, the existing table will be passed back out, otherwise, the argument `tb` is set to null. In either case, the caller can call `delete` on the variable and should do so to avoid memory leak.

3.101.2.2 iterator `ibis::tableList::end () const` `[inline]`

Return the iterator to the end of the list.

Following STL convention, the `end` is always one past the last element in the list.

3.101.2.3 `[] const ibis::table* ibis::tableList::operator[] (const char * tname) const` `[inline]`

Find the named table.

Returns null pointer if no table with the given name is found.

3.101.2.4 void `ibis::tableList::remove (const char * tname)` `[inline]`

Remove the named data table from the list.

The destructor of this function automatically clean up all table objects, there is no need to explicit remove them.

The documentation for this class was generated from the following file:

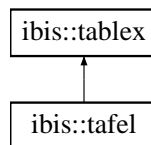
- [table.h](#)

3.102 `ibis::tablex` Class Reference

The class for expandable tables.

```
#include <table.h>
```

Inheritance diagram for `ibis::tablex`:

**Public Member Functions**

- virtual int **addColumn** (const char **cname*, `ibis::TYPE_T` *ctype*, const char **cdesc*)=0
- virtual int **addColumn** (const char **cname*, `ibis::TYPE_T` *ctype*)=0
Add a column.
- virtual int **append** (const char **cname*, `uint64_t` *begin*, `uint64_t` *end*, void **values*)=0
Add values to the named column.
- virtual int **appendRow** (const char **line*, const char **delimiters*=0)=0
Append a row stored in ASCII form.
- virtual int **appendRow** (const `ibis::table::row` &)=0
Add one row.
- virtual int **appendRows** (const `std::vector< ibis::table::row >` &)=0
Add multiple rows.
- virtual int **readCSV** (const char **filename*, const char **delimiters*=0)=0
Read the content of the specified as comma-separated values.

- virtual int `write` (const char *dir, const char *tname, const char *tdesc) const =0
Write the in-memory data records to the specified directory on disk.

Static Public Member Functions

- static `ibis::tablex * create` ()
Create a minimalistic table exclusively for entering new records.
- static `ibis::tablex * makeExtensible` (`ibis::table *t`)
Make the incoming table expandable. Not yet implemented.

3.102.1 Detailed Description

The class for expandable tables.

Note:

Each function that returns an integer returns 0 in case of success, a negative value in case error and a positive number as advisory information.

3.102.2 Member Function Documentation

3.102.2.1 virtual int `ibis::tablex::append` (const char * *cname*, uint64_t *begin*, uint64_t *end*, void * *values*) [pure virtual]

Add values to the named column.

The column name must be in the table already. The first value is to be placed at row `begin` (the row numbers start with 0) and the last value before row `end`. The array `values` must contain values of the correct type corresponding to the type specified before.

Note:

Since each column may have different number of rows filled, the number of rows in the table is considered to be the maximum number of rows filled of all columns.

This function can not be used to introduce new columns in a table. A new column must be added with `addColumn`.

`appendRow`

Implemented in `ibis::tafel`.

3.102.2.2 virtual int `ibis::tablex::appendRow` (const char * *line*, const char * *delimiters* = 0) [pure virtual]

Append a row stored in ASCII form.

The ASCII form of the values are assumed to be separated by comma (,) or space, but additional delimiters may be added through the second argument.

Implemented in `ibis::tafel`.

3.102.2.3 `virtual int ibis::tablex::appendRow (const ibis::table::row &) [pure virtual]`

Add one row.

If an array of names has the same number of elements as the array of values, the names are used as column names. If the names are not specified explicitly, the values are assigned to the columns of the same data type in the order as they are specified through `addColumn` or if the same order as they are recreated from an existing dataset (which is typically alphabetical).

Note:

The column names are not case-sensitive.

Like `append`, this function can not be used to introduce new columns in a table. A new column must be added with `addColumn`.

Since the various columns may have different numbers of rows filled, the number of rows in the table is assumed to the largest number of rows filled so far. The new row appended here increases the number of rows in the table by 1. The unfilled rows are assumed to be null.

A null value of an integer column is recorded as the maximum possible of the type of integer. A null value of a floating-point valued column is recorded as a quiet NaN (Not-a-Number). A null value of a string-valued column is recorded as an empty string. In all cases, a null mask is used to indicate that they are null values.

Implemented in [ibis::tafel](#).

3.102.2.4 `virtual int ibis::tablex::appendRows (const std::vector< ibis::table::row > &) [pure virtual]`

Add multiple rows.

Rows in the incoming vector are processed one after another. The ordering of the values in earlier rows are automatically carried over to the later rows until another set of names is specified. [appendRow](#)

Implemented in [ibis::tafel](#).

3.102.2.5 `virtual int ibis::tablex::readCSV (const char * filename, const char * delimiters = 0) [pure virtual]`

Read the content of the specified as comma-separated values.

Append the records to this table. By default the records are delimited by comma (,) and blank space. One may specify additional delimiters using the second argument.

Implemented in [ibis::tafel](#).

3.102.2.6 `virtual int ibis::tablex::write (const char * dir, const char * tname, const char * tdesc) const [pure virtual]`

Write the in-memory data records to the specified directory on disk.

If the table name (`tname`) is a null string or an empty, the last component of the directory name is used. If the description (`tdesc`) is a null string or an empty string, a time stamp will be printed in its place. If the specified directory already contains data, the new records will be appended to the existing data. In this case, the table name specified here will overwrite the existing name, but the existing name and description will be retained if the current arguments are null strings or empty strings. The data type associated with this table will overwrite the existing data type information.

Implemented in [ibis::tafel](#).

The documentation for this class was generated from the following files:

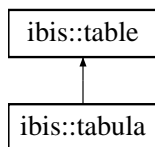
- [table.h](#)
- [tafel.cpp](#)

3.103 `ibis::tabula` Class Reference

A trivial class for table with no columns.

```
#include <tab.h>
```

Inheritance diagram for `ibis::tabula`:



Public Member Functions

- virtual int [buildIndex](#) (const char *, const char *)
Create the index for named column.
- virtual int [buildIndexes](#) (const char *)
Create indexes for every column of the table.
- virtual [stringList columnNames](#) () const
Return column names.
- virtual [typeList columnTypes](#) () const
Return data types.
- virtual `ibis::table::cursor *` [createCursor](#) () const
Create a `CURSOR` object to perform row-wise data access.
- virtual void [describe](#) (std::ostream &) const
Print a description of the table to the specified output stream.
- virtual int [dump](#) (std::ostream &, const char *) const
Dump the values in ASCII form to the specified output stream.
- virtual void [estimate](#) (const char *cond, uint64_t &nmin, uint64_t &nmax) const
Estimate the number of rows satisfying the selection conditions.
- virtual int64_t [getColumnAsBytes](#) (const char *, char *) const
Retrieve all values of the named column.
- virtual int64_t [getColumnAsDoubles](#) (const char *, double *) const
Retrieve all values of the named column.
- virtual int64_t [getColumnAsFloats](#) (const char *, float *) const
Retrieve all values of the named column.
- virtual int64_t [getColumnAsInts](#) (const char *, int32_t *) const
Retrieve all values of the named column.
- virtual int64_t [getColumnAsLongs](#) (const char *, int64_t *) const
Retrieve all values of the named column.

- virtual int64_t [getColumnAsShorts](#) (const char *, int16_t *) const
Retrieve all values of the named column.
- virtual int64_t [getColumnAsStrings](#) (const char *, std::vector< std::string > &) const
Retrieve the null-terminated strings as a vector of std::string objects.
- virtual int64_t [getColumnAsUBytes](#) (const char *, unsigned char *) const
Retrieve all values of the named column.
- virtual int64_t [getColumnAsUInts](#) (const char *, uint32_t *) const
Retrieve all values of the named column.
- virtual int64_t [getColumnAsULongs](#) (const char *, uint64_t *) const
Retrieve all values of the named column.
- virtual int64_t [getColumnAsUShorts](#) (const char *, uint16_t *) const
Retrieve all values of the named column.
- virtual long [getHistogram](#) (const char *, const char *, double, double, double, std::vector< size_t > &) const
Compute the histogram of the named column.
- virtual long [getHistogram2D](#) (const char *, const char *, double, double, double, const char *, double, double, double, std::vector< size_t > &) const
Compute a two-dimension histogram on columns `cname1` and `name2`.
- virtual long [getHistogram3D](#) (const char *, const char *, double, double, double, const char *, double, double, double, double, const char *, double, double, double, std::vector< size_t > &) const
Compute a three-dimensional histogram on the named columns.
- virtual **table** * [groupby](#) (const [stringList](#) &) const
Perform aggregate functions on the current table.
- virtual void [indexSpec](#) (const char *, const char *)
Replace the current indexing option.
- virtual const char * [indexSpec](#) (const char *) const
Retrieve the current indexing option.
- virtual size_t [nColumns](#) () const
- virtual uint64_t [nRows](#) () const
- virtual void [orderby](#) (const [stringList](#) &)
Reorder the rows.
- virtual void [reverseRows](#) ()
Reverse the order of the rows.
- virtual **table** * [select](#) (const char *, const char *) const
Given a set of column names and a set of selection conditions, compute another table that represents the selected values.
- **tabula** (size_t nr=0)
- **tabula** (const char *na, const char *de, uint64_t nr)

Classes

- class `cursor`

3.103.1 Detailed Description

A trivial class for table with no columns.

This type of table is generated when the select clause is blank or not specified.

3.103.2 Member Function Documentation

3.103.2.1 `virtual int ibis::tabula::buildIndex (const char *, const char *) [inline, virtual]`

Create the index for named column.

The existing index will be replaced. If an indexing option is not specified, it will use the internally recorded option for the named column or the table containing the column.

Note:

Unless any there is a specific instruction to not index a column, the querying functions will automatically build indices as necessary. However, as building an index is relatively expensive process, building an index on a column is on average about four or five times as expensive as reading the column from disk, this function is provided so that it is possible to build indexes beforehand.

Implements [ibis::table](#).

3.103.2.2 `virtual int ibis::tabula::buildIndexes (const char *) [inline, virtual]`

Create indexes for every column of the table.

Existing indexes will be replaced. If an indexing option is not specified, the internally recorded options will be used. [buildIndex](#)

Implements [ibis::table](#).

3.103.2.3 `virtual int ibis::tabula::dump (std::ostream &, const char *) const [inline, virtual]`

Dump the values in ASCII form to the specified output stream.

The default delimiter is coma (","), which produces Comma-Separated-Values (CSV).

Implements [ibis::table](#).

3.103.2.4 `void ibis::tabula::estimate (const char * cond, uint64_t & nmin, uint64_t & nmax) const [inline, virtual]`

Estimate the number of rows satisfying the selection conditions.

The number of rows is between [nmin, nmax].

Implements [ibis::table](#).

3.103.2.5 `virtual int64_t ibis::tabula::getColumnAsBytes (const char *, char *) const [inline, virtual]`

Retrieve all values of the named column.

The member functions of this class only support access to whole column at a time. Use `table::cursor` class for row-wise accesses. For fixed-width data types, the raw pointers are used to point to the values to be returned. In these cases, the caller is responsible for allocating enough storage for the values to be returned.

Implements [ibis::table](#).

3.103.2.6 `virtual int64_t ibis::tabula::getColumnAsDoubles (const char *, double *) const` [inline, virtual]

Retrieve all values of the named column.

The member functions of this class only support access to whole column at a time. Use `table::cursor` class for row-wise accesses. For fixed-width data types, the raw pointers are used to point to the values to be returned. In these cases, the caller is responsible for allocating enough storage for the values to be returned.

Implements [ibis::table](#).

3.103.2.7 `virtual int64_t ibis::tabula::getColumnAsFloats (const char *, float *) const` [inline, virtual]

Retrieve all values of the named column.

The member functions of this class only support access to whole column at a time. Use `table::cursor` class for row-wise accesses. For fixed-width data types, the raw pointers are used to point to the values to be returned. In these cases, the caller is responsible for allocating enough storage for the values to be returned.

Implements [ibis::table](#).

3.103.2.8 `virtual int64_t ibis::tabula::getColumnAsInts (const char *, int32_t *) const` [inline, virtual]

Retrieve all values of the named column.

The member functions of this class only support access to whole column at a time. Use `table::cursor` class for row-wise accesses. For fixed-width data types, the raw pointers are used to point to the values to be returned. In these cases, the caller is responsible for allocating enough storage for the values to be returned.

Implements [ibis::table](#).

3.103.2.9 `virtual int64_t ibis::tabula::getColumnAsLongs (const char *, int64_t *) const` [inline, virtual]

Retrieve all values of the named column.

The member functions of this class only support access to whole column at a time. Use `table::cursor` class for row-wise accesses. For fixed-width data types, the raw pointers are used to point to the values to be returned. In these cases, the caller is responsible for allocating enough storage for the values to be returned.

Implements [ibis::table](#).

3.103.2.10 `virtual int64_t ibis::tabula::getColumnAsShorts (const char *, int16_t *) const` [inline, virtual]

Retrieve all values of the named column.

The member functions of this class only support access to whole column at a time. Use `table::cursor` class for row-wise accesses. For fixed-width data types, the raw pointers are used to point to the values to be returned. In these cases, the caller is responsible for allocating enough storage for the values to be returned.

Implements [ibis::table](#).

3.103.2.11 `virtual int64_t ibis::tabula::getColumnAsStrings (const char *, std::vector< std::string > &) const` [inline, virtual]

Retrieve the null-terminated strings as a vector of `std::string` objects.

Both `ibis::CATEGORY` and `ibis::TEXT` types can be retrieved using this function.

Implements `ibis::table`.

3.103.2.12 `virtual int64_t ibis::tabula::getColumnAsUBytes (const char *, unsigned char *) const` [inline, virtual]

Retrieve all values of the named column.

The member functions of this class only support access to whole column at a time. Use `table::cursor` class for row-wise accesses. For fixed-width data types, the raw pointers are used to point to the values to be returned. In these cases, the caller is responsible for allocating enough storage for the values to be returned.

Implements `ibis::table`.

3.103.2.13 `virtual int64_t ibis::tabula::getColumnAsUInts (const char *, uint32_t *) const` [inline, virtual]

Retrieve all values of the named column.

The member functions of this class only support access to whole column at a time. Use `table::cursor` class for row-wise accesses. For fixed-width data types, the raw pointers are used to point to the values to be returned. In these cases, the caller is responsible for allocating enough storage for the values to be returned.

Implements `ibis::table`.

3.103.2.14 `virtual int64_t ibis::tabula::getColumnAsULongs (const char *, uint64_t *) const` [inline, virtual]

Retrieve all values of the named column.

The member functions of this class only support access to whole column at a time. Use `table::cursor` class for row-wise accesses. For fixed-width data types, the raw pointers are used to point to the values to be returned. In these cases, the caller is responsible for allocating enough storage for the values to be returned.

Implements `ibis::table`.

3.103.2.15 `virtual int64_t ibis::tabula::getColumnAsUShorts (const char *, uint16_t *) const` [inline, virtual]

Retrieve all values of the named column.

The member functions of this class only support access to whole column at a time. Use `table::cursor` class for row-wise accesses. For fixed-width data types, the raw pointers are used to point to the values to be returned. In these cases, the caller is responsible for allocating enough storage for the values to be returned.

Implements `ibis::table`.

3.103.2.16 `virtual long ibis::tabula::getHistogram (const char *, const char *, double, double, double, std::vector< size_t > &) const` [inline, virtual]

Compute the histogram of the named column.

This version uses the user specified bins: [begin, begin+stride) [begin+stride, begin+2*stride), A record is placed in bin

$(x - \text{begin}) / \text{stride}$, where the first bin is bin 0. This gives a total of

$(\text{end} - \text{begin}) / \text{stride}$ bins.

Note:

Records (rows) outside of the range `[begin, end]` are not counted.

Non-positive `stride` is considered as an error.

If `end` is less than `begin`, an empty array `counts` is returned along with return value 0.

Implements [`ibis::table`](#).

3.103.2.17 `virtual long ibis::tabula::getHistogram2D (const char *, const char *, double, double, double, const char *, double, double, double, std::vector< size_t > &) const` `[inline, virtual]`

Compute a two-dimension histogram on columns `cname1` and `name2`.

The bins along each dimension are defined the same way as in function [`getHistogram`](#). The array `counts` stores the two-dimensional bins with the first dimension as the slow varying dimension following C convention for ordering multi-dimensional arrays.

Implements [`ibis::table`](#).

3.103.2.18 `virtual long ibis::tabula::getHistogram3D (const char *, const char *, double, double, double, const char *, double, double, double, const char *, double, double, double, std::vector< size_t > &) const` `[inline, virtual]`

Compute a three-dimensional histogram on the named columns.

The triplets `<begin, end, stride>` are used the same ways in [`getHistogram`](#) and [`getHistogram2D`](#). The three dimensional bins are linearized in `counts` with the first being the slowest varying dimension and the third being the fastest varying dimension following the C convention for ordering multi-dimensional arrays.

Implements [`ibis::table`](#).

3.103.2.19 `virtual table* ibis::tabula::groupby (const stringList &) const` `[inline, virtual]`

Perform aggregate functions on the current table.

It produces a new table. The list of strings passed to this function are interpreted as a set of names followed by a set of functions. Currently, only functions COUNT, AVG, MIN, MAX, and SUM are supported.

Implements [`ibis::table`](#).

3.103.2.20 `virtual void ibis::tabula::indexSpec (const char *, const char *)` `[inline, virtual]`

Replace the current indexing option.

If no column name is specified, it resets the indexing option for the table.

Implements [`ibis::table`](#).

3.103.2.21 `virtual const char* ibis::tabula::indexSpec (const char *) const` `[inline, virtual]`

Retrieve the current indexing option.

If no column name is specified, it retrieve the indexing option for the table.

Implements [`ibis::table`](#).

3.103.2.22 `virtual void ibis::tabula::orderby (const stringList &)` `[inline, virtual]`

Reorder the rows.

Sort the rows in ascending order of the columns specified in the list of column names. This function is not designated `const` because though it does not change the content in SQL logic, but it may change internal representations.

Note:

If an empty list is passed to this function, it will reorder rows using all columns with the column having the smallest number of distinct values first.

Implements [ibis::table](#).

The documentation for this class was generated from the following file:

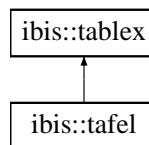
- [tab.h](#)

3.104 `ibis::tafel` Class Reference

An expandable table.

```
#include <tafel.h>
```

Inheritance diagram for `ibis::tafel::`

**Public Member Functions**

- virtual int **addColumn** (const char *cname, [ibis::TYPE_T](#) ctype, const char *cdesc)
- virtual int **addColumn** (const char *cname, [ibis::TYPE_T](#) ctype)
Add a column.
- virtual int **append** (const char *cname, uint64_t begin, uint64_t end, void *values)
Add values to the named column.
- virtual int **appendRow** (const char *, const char *)
Append a row stored in ASCII form.
- virtual int **appendRow** (const [ibis::table::row](#) &)
Add one row.
- virtual int **appendRows** (const std::vector< [ibis::table::row](#) > &)
Add multiple rows.
- virtual int **readCSV** (const char *filename, const char *delimiters)
Read the content of the specified as comma-separated values.
- virtual int **write** (const char *dir, const char *tname, const char *tdesc) const
Write the in-memory data records to the specified directory on disk.

Protected Types

- typedef std::map< const char *, column *, [ibis::lessi](#) > **columnList**

Protected Member Functions

- `template<typename T> void append (const std::vector< std::string > &nm, const std::vector< T > &va, std::vector< array_t< T > * > &buf)`
- `template<typename T> void append (const T *in, ibis::bitvector::word_t be, ibis::bitvector::word_t en, array_t< T > &out, const T &fill, ibis::bitvector &mask) const`
- `void appendString (const std::vector< std::string > &nm, const std::vector< std::string > &va, std::vector< std::vector< std::string > * > &buf)`
- `void appendString (const std::vector< std::string > *in, ibis::bitvector::word_t be, ibis::bitvector::word_t en, std::vector< std::string > &out, ibis::bitvector &mask) const`
- `void clear ()`
Clear the content of the buffers.
- `template<typename T> void locate (ibis::TYPE_T, std::vector< array_t< T > * > &buf) const`
- `void locateString (ibis::TYPE_T t, std::vector< std::vector< std::string > * > &buf) const`
- `void normalize ()`
Make all short columns catch up with the longest one.
- `int parseLine (const char *str, const char *del, const char *id)`
- `template<typename T> int writeColumn (int fdes, ibis::bitvector::word_t nold, ibis::bitvector::word_t nnew, const array_t< T > &vals, const T &fill, ibis::bitvector &totmask, const ibis::bitvector &newmask) const`
- `int writeString (int fdes, ibis::bitvector::word_t nold, ibis::bitvector::word_t nnew, const std::vector< std::string > &vals, ibis::bitvector &totmask, const ibis::bitvector &newmask) const`

Protected Attributes

- `std::vector< column * > colorder`
Order of columns as they were specified through `addColumn`.
- `columnList cols`
List of columns in alphabetical order.
- `ibis::bitvector::word_t nrows`
Number of rows of this table.

Classes

- struct **column**

3.104.1 Detailed Description

An expandable table.

It inherits from [ibis::tablex](#) only therefore does not support any querying functions.

Note:

The word `tafel` is a German word for table.

3.104.2 Member Function Documentation

3.104.2.1 `int ibis::tafel::append (const char * cname, uint64_t begin, uint64_t end, void * values)` [virtual]

Add values to the named column.

The column name must be in the table already. The first value is to be placed at row `begin` (the row numbers start with 0) and the last value before row `end`. The array `values` must contain values of the correct type corresponding to the type specified before.

Note:

Since each column may have different number of rows filled, the number of rows in the table is considered to be the maximum number of rows filled of all columns.

This function can not be used to introduce new columns in a table. A new column must be added with `addColumn`.

[appendRow](#)

Implements [ibis::tablex](#).

3.104.2.2 `int ibis::tafel::appendRow (const char *, const char *)` [virtual]

Append a row stored in ASCII form.

The ASCII form of the values are assumed to be separated by comma (,) or space, but additional delimiters may be added through the second argument.

Implements [ibis::tablex](#).

3.104.2.3 `int ibis::tafel::appendRow (const ibis::table::row &)` [virtual]

Add one row.

If an array of names has the same number of elements as the array of values, the names are used as column names. If the names are not specified explicitly, the values are assigned to the columns of the same data type in the order as they are specified through `addColumn` or if the same order as they are recreated from an existing dataset (which is typically alphabetical).

Note:

The column names are not case-sensitive.

Like `append`, this function can not be used to introduce new columns in a table. A new column must be added with `addColumn`.

Since the various columns may have different numbers of rows filled, the number of rows in the table is assumed to the largest number of rows filled so far. The new row appended here increases the number of rows in the table by 1. The unfilled rows are assumed to be null.

A null value of an integer column is recorded as the maximum possible of the type of integer. A null value of a floating-point valued column is recorded as a quiet NaN (Not-a-Number). A null value of a string-valued column is recorded as an empty string. In all cases, a null mask is used to indicate that they are null values.

Implements [ibis::tablex](#).

3.104.2.4 `int ibis::tafel::appendRows (const std::vector< ibis::table::row > &)` [virtual]

Add multiple rows.

Rows in the incoming vector are processed one after another. The ordering of the values in earlier rows are automatically carried over to the later rows until another set of names is specified. [appendRow](#)

Implements [ibis::tablex](#).

3.104.2.5 `int ibis::tafel::readCSV (const char *filename, const char *delimiters)` [virtual]

Read the content of the specified as comma-separated values.

Append the records to this table. By default the records are delimited by comma (,) and blank space. One may specify additional delimiters using the second argument.

Implements [ibis::tablex](#).

3.104.2.6 `int ibis::tafel::write (const char *dir, const char *tname, const char *tdesc) const` [virtual]

Write the in-memory data records to the specified directory on disk.

If the table name (`tname`) is a null string or an empty, the last component of the directory name is used. If the description (`tdesc`) is a null string or an empty string, a time stamp will be printed in its place. If the specified directory already contains data, the new records will be appended to the existing data. In this case, the table name specified here will overwrite the existing name, but the existing name and description will be retained if the current arguments are null strings or empty strings. The data type associated with this table will overwrite the existing data type information.

Implements [ibis::tablex](#).

The documentation for this class was generated from the following files:

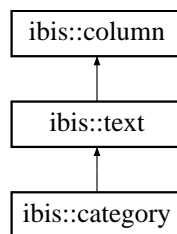
- [tafel.h](#)
- [tafel.cpp](#)

3.105 `ibis::text` Class Reference

A minimalistic structure for storing arbitrary text fields.

```
#include <category.h>
```

Inheritance diagram for `ibis::text`:

**Public Member Functions**

- virtual long [append](#) (const char *dt, const char *df, const uint32_t nold, const uint32_t nnew, const uint32_t nbuf, char *buf)

Append the data file stored in directory `df` to the corresponding data file in directory `dt`.

- virtual double [estimateCost](#) (const [ibis::qMultiString](#) &cmp) const
- virtual double [estimateCost](#) (const [ibis::qString](#) &cmp) const
- virtual const char * [findString](#) (const char *str) const

If the input string is found in the data file, it is returned, else this function returns 0.

- virtual void [getString](#) (uint32_t i, std::string &val) const
- const [column](#) * [IDColumnForKeywordIndex](#) () const
- virtual long [keywordSearch](#) (const char *str) const

- virtual long `keywordSearch` (const char *str, `ibis::bitvector` &hits) const
- virtual void `print` (std::ostream &out) const
- virtual long `search` (const std::vector< std::string > &strs) const
- virtual long `search` (const char *str) const
- virtual long `search` (const std::vector< std::string > &strs, `ibis::bitvector` &hits) const
Given a group of string literals, return a bitvector that matches anyone of the input strings.
- virtual long `search` (const char *str, `ibis::bitvector` &hits) const
Given a string literal, return a bitvector that marks the strings that matches it.
- virtual std::vector< std::string > * `selectStrings` (const `bitvector` &mask) const
- virtual `array_t`< uint32_t > * `selectUInts` (const `bitvector` &mask) const
Return the integer values of the records marked 1 in the mask.
- `text` (const `ibis::column` &col)
- `text` (const `part` *tbl, const char *name, `ibis::TYPE_T` t=`ibis::TEXT`)
- `text` (const `part` *tbl, FILE *file)
- virtual void `write` (FILE *file) const
Write the current content to the TDC file.

Protected Member Functions

- int `readString` (std::string &, int, long, long, char *, uint32_t, uint32_t &, off_t &) const
Read on string from an open file.
- void `readString` (uint32_t i, std::string &val) const
Return the ith string, i.e., the string in the ith row/record.
- void `startPositions` (const char *dir, uint32_t nbuf, char *buf) const
Locate the starting position of each string and write the positions as unsigned integers to a file with .sp as extension.

3.105.1 Detailed Description

A minimalistic structure for storing arbitrary text fields.

The keyword search operation is implemented through a boolean term-document matrix (`ibis::keywords`) that is actually generated externally.

3.105.2 Member Function Documentation

3.105.2.1 long `ibis::text::append` (const char * dt, const char * df, const uint32_t nold, const uint32_t nnew, const uint32_t nbuf, char * buf) [virtual]

Append the data file stored in directory df to the corresponding data file in directory dt.

Use the buffer buf to copy data in large chunks.

Note:

No error checking is performed.

Does not check for missing entries. May cause records to be misaligned.

Reimplemented from `ibis::column`.

Reimplemented in `ibis::category`.

3.105.2.2 `const char * ibis::text::findString (const char * str) const` [virtual]

If the input string is found in the data file, it is returned, else this function returns 0.

It needs to keep both the data file and the starting position file open at the same time.

Reimplemented from [ibis::column](#).

3.105.2.3 `int ibis::text::readString (std::string & res, int fdes, long be, long en, char * buf, uint32_t nbuf, uint32_t & inbuf, off_t & boffset) const` [protected]

Read on string from an open file.

The string starts at position *be* and ends at *en*. The content may be in the array *buf*.

3.105.2.4 `void ibis::text::readString (uint32_t i, std::string & ret) const` [protected]

Return the *i*th string, i.e., the string in the *i*th row/record.

It goes through a two-stage process by reading from two files. This is quite slow!

3.105.2.5 `array_t< uint32_t > * ibis::text::selectUInts (const bitvector & mask) const` [virtual]

Return the integer values of the records marked 1 in the mask.

This indicates to [ibis::bundle](#) that every string value is distinct. It also forces the sorting procedure to produce an order following the order of the entries in the table. This makes the print out of an [ibis::text](#) field quite less useful than others!

Reimplemented from [ibis::column](#).

Reimplemented in [ibis::category](#).

3.105.2.6 `void ibis::text::startPositions (const char * dir, uint32_t nbuf, char * buf) const` [protected]

Locate the starting position of each string and write the positions as unsigned integers to a file with `.sp` as extension.

If *dir* is a nil pointer, the directory is default to the current working directory of the data table. Arguments *nbuf* and *buf* are used as temporary working space. If *nbuf* = 0, this function allocates its own working space.

The documentation for this class was generated from the following files:

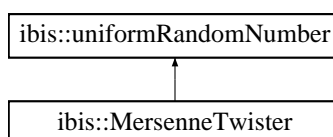
- [category.h](#)
- [category.cpp](#)

3.106 `ibis::uniformRandomNumber` Class Reference

A functor to generate uniform random number in the range [0, 1).

```
#include <twister.h>
```

Inheritance diagram for `ibis::uniformRandomNumber`:



Public Member Functions

- virtual double `operator() ()=0`

3.106.1 Detailed Description

A functor to generate uniform random number in the range [0, 1).

The documentation for this class was generated from the following file:

- [twister.h](#)

3.107 `ibis::util::counter` Class Reference

A simple global counter.

```
#include <util.h>
```

Public Member Functions

- `counter` (const char *m="ibis::util::counter")
- uint32_t `operator() ()`
Return the current count and increment the count.
- void `reset ()`
Reset count to zero.
- uint32_t `value () const`
Return the current count value.

3.107.1 Detailed Description

A simple global counter.

Each time the `operator()` is called, it is incremented by 1. Calls from different threads are serialized through a mutual exclusion lock.

The documentation for this class was generated from the following file:

- [util.h](#)

3.108 `ibis::util::ioLock` Class Reference

A global IO mutex lock.

```
#include <util.h>
```

3.108.1 Detailed Description

A global IO mutex lock.

The documentation for this class was generated from the following files:

- [util.h](#)
- [util.cpp](#)

3.109 `ibis::util::mutexLock` Class Reference

An wrapper class for perform `pthread_mutex_lock/unlock`.

```
#include <util.h>
```

Public Member Functions

- **mutexLock** (`pthread_mutex_t *lk`, `const char *m`)

3.109.1 Detailed Description

An wrapper class for perform `pthread_mutex_lock/unlock`.

The documentation for this class was generated from the following file:

- [util.h](#)

3.110 `ibis::util::quietLock` Class Reference

An wrapper class for perform `pthread_mutex_lock/unlock`.

```
#include <util.h>
```

Public Member Functions

- **quietLock** (`pthread_mutex_t *lk`, `const char *m`)

3.110.1 Detailed Description

An wrapper class for perform `pthread_mutex_lock/unlock`.

Avoid invoking `ibis::util::logMessage` so it can be used inside `ibis::util::logMessage`.

The documentation for this class was generated from the following file:

- [util.h](#)

3.111 `ibis::util::readLock` Class Reference

An wrapper class for perform `pthread_rwlock_rdlock/unlock`.

```
#include <util.h>
```

Public Member Functions

- **readLock** (`pthread_rwlock_t *lk`, `const char *m`)

3.111.1 Detailed Description

An wrapper class for perform `pthread_rwlock_rdlock/unlock`.

The documentation for this class was generated from the following file:

- [util.h](#)

3.112 `ibis::util::writeLock` Class Reference

An wrapper class for perform `pthread_rwlock_wrlock/unlock`.

```
#include <util.h>
```

Public Member Functions

- **`writeLock`** (`pthread_rwlock_t *lk`, `const char *m`)

3.112.1 Detailed Description

An wrapper class for perform `pthread_rwlock_wrlock/unlock`.

The documentation for this class was generated from the following file:

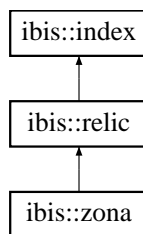
- [util.h](#)

3.113 `ibis::zona` Class Reference

The two-level equality-equality code.

```
#include <irelic.h>
```

Inheritance diagram for `ibis::zona`:



Public Member Functions

- virtual long **`append`** (`const char *dt`, `const char *df`, `uint32_t nnew`)
Extend the index.
- virtual `uint32_t` **`estimate`** (`const ibis::qContinuousRange &expr`) `const`
Returns an upper bound on the number of hits.
- virtual `double` **`estimateCost`** (`const ibis::qContinuousRange &expr`) `const`
Estimate the code of evaluate a range condition.
- virtual long **`evaluate`** (`const ibis::qContinuousRange &expr`, `ibis::bitvector &hits`) `const`
To evaluate the exact hits.
- virtual `const char *` **`name`** () `const`
Returns the name of the index, similar to the function `type`, but returns a string instead.
- virtual void **`print`** (`std::ostream &out`) `const`
Prints human readable information.

- virtual void `read` (`ibis::fileManager::storage *st`)
Reconstructs an index from an array of bytes.
- virtual void `read` (`const char *idxfile`)
Reconstructs an index from the named file.
- virtual `INDEX_TYPE type` () const
Returns an index type identifier.
- virtual void `write` (`const char *dt`) const
Save index to a file.
- `zona` (`const ibis::column *c`, `ibis::fileManager::storage *st`, `uint32_t offset=8`)
The leading portion of the index file is the same as `ibis::relic`, which allows the constructor of the base class to work properly.
- `zona` (`const ibis::column *c=0`, `const char *f=0`)

Protected Member Functions

- virtual void `clear` ()
Clear the existing content.

3.113.1 Detailed Description

The two-level equality-equality code.

Note:

zone is Italian word for zone, the name of the binned version of the two-level equality-equality code.

3.113.2 Constructor & Destructor Documentation

3.113.2.1 `ibis::zona::zona` (`const ibis::column *c`, `ibis::fileManager::storage *st`, `uint32_t start = 8`)

The leading portion of the index file is the same as `ibis::relic`, which allows the constructor of the base class to work properly.

The content following the last bitvector in `ibis::relic` is as follows, writeCoarse.

`nc` (`uint32_t`) – number of coarse bins. `cbounds` (`unsigned[nc+1]`) – boundaries of the coarse bins. `coffsets`(`int32_t[nc+1]`) – starting position of the coarse level bitmaps. `cbits` (`bitvector[nc]`) – bitvector laid out one after another.

3.113.3 Member Function Documentation

3.113.3.1 `long ibis::zona::evaluate` (`const ibis::qContinuousRange & expr`, `ibis::bitvector & hits`) const [virtual]

To evaluate the exact hits.

On success, return the number of hits, otherwise a negative value is returned.

Reimplemented from `ibis::relic`.

3.113.3.2 `void ibis::zona::print (std::ostream & out) const` [virtual]

Prints human readable information.

Outputs information about the index as text to the specified output stream.

Reimplemented from [ibis::relic](#).

3.113.3.3 `void ibis::zona::read (ibis::fileManager::storage * st)` [virtual]

Reconstructs an index from an array of bytes.

Intended for internal use only!

Reimplemented from [ibis::relic](#).

3.113.3.4 `void ibis::zona::read (const char * idxfile)` [virtual]

Reconstructs an index from the named file.

The name can be the directory containing an index file. In this case, the name of the index file must be the name of the column followed by ".idx" suffix.

Reimplemented from [ibis::relic](#).

3.113.3.5 `void ibis::zona::write (const char * dt) const` [virtual]

Save index to a file.

Outputs the index in a compact binary format to the named file or directory. The index file contains a header that can be identified by the function `isIndex`.

Reimplemented from [ibis::relic](#).

The documentation for this class was generated from the following files:

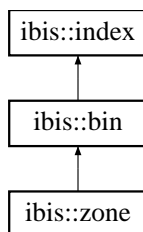
- [irelic.h](#)
- [ixzona.cpp](#)

3.114 `ibis::zone` Class Reference

A two-level index.

```
#include <ibin.h>
```

Inheritance diagram for `ibis::zone`:

**Public Member Functions**

- virtual void **adjustLength** (uint32_t nrows)
- long **append** (const [ibis::zone](#) &tail)
- virtual long **append** (const char *dt, const char *df, uint32_t nnew)

Extend the index.

- virtual void `binBoundaries` (`std::vector< double > &`) const
The function `binBoundaries` and `binWeights` return bin boundaries and counts of each bin respectively.
- virtual void `binWeights` (`std::vector< uint32_t > &`) const
- virtual void `estimate` (const `ibis::qContinuousRange` &expr, `ibis::bitvector` &lower, `ibis::bitvector` &upper) const
Computes an approximation of hits as a pair of lower and upper bounds.
- virtual long `evaluate` (const `ibis::qContinuousRange` &expr, `ibis::bitvector` &hits) const
To evaluate the exact hits.
- virtual const char * `name` () const
Returns the name of the index, similar to the function `type`, but returns a string instead.
- virtual `uint32_t numBins` () const
- virtual void `print` (`std::ostream &out`) const
Prints human readable information.
- virtual void `read` (`ibis::fileManager::storage` *st)
Reconstructs an index from an array of bytes.
- virtual void `read` (const char *idxfile)
Reconstructs an index from the named file.
- virtual void `speedTest` (`std::ostream &out`) const
Time some logical operations and print out their speed.
- virtual `INDEX_TYPE type` () const
Returns an index type identifier.
- virtual float `undecidable` (const `ibis::qContinuousRange` &expr, `ibis::bitvector` &iffy) const
Mark the position of the rows that can not be decided with this index.
- virtual void `write` (const char *dt) const
Save index to a file.
- `zone` (const `ibis::bin` &rhs)
- `zone` (const `ibis::column` *c, `ibis::fileManager::storage` *st, `uint32_t` offset=8)

3.114.1 Detailed Description

A two-level index.

Both levels are not cumulative, i.e., both levels are equality encoded.

3.114.2 Member Function Documentation

3.114.2.1 void `ibis::zone::estimate` (const `ibis::qContinuousRange` & expr, `ibis::bitvector` & lower, `ibis::bitvector` & upper) const `[virtual]`

Computes an approximation of hits as a pair of lower and upper bounds.

Parameters:

expr the query expression to be evaluated.

lower a bitvector marking a subset of the hits. All rows marked with one (1) are definitely hits.

upper a bitvector marking a superset of the hits. All hits are marked with one, but some of the rows marked one may not be hits. If the variable *upper* is empty, the variable *lower* is assumed to contain the exact answer.

Reimplemented from [ibis::bin](#).

3.114.2.2 `long ibis::zone::evaluate (const ibis::qContinuousRange & expr, ibis::bitvector & hits) const` [virtual]

To evaluate the exact hits.

On success, return the number of hits, otherwise a negative value is returned.

Reimplemented from [ibis::bin](#).

3.114.2.3 `void ibis::zone::print (std::ostream & out) const` [virtual]

Prints human readable information.

Outputs information about the index as text to the specified output stream.

Reimplemented from [ibis::bin](#).

3.114.2.4 `void ibis::zone::read (ibis::fileManager::storage * st)` [virtual]

Reconstructs an index from an array of bytes.

Intended for internal use only!

Reimplemented from [ibis::bin](#).

3.114.2.5 `void ibis::zone::read (const char * idxfile)` [virtual]

Reconstructs an index from the named file.

The name can be the directory containing an index file. In this case, the name of the index file must be the name of the column followed by ".idx" suffix.

Reimplemented from [ibis::bin](#).

3.114.2.6 `float ibis::zone::undecidable (const ibis::qContinuousRange & expr, ibis::bitvector & iffy) const` [virtual]

Mark the position of the rows that can not be decided with this index.

Parameters:

expr the range conditions to be evaluated.

iffy the bitvector marking the positions of rows that can not be decided using the index. Return value is the expected fraction of undecided rows that might satisfy the range conditions.

Reimplemented from [ibis::bin](#).

3.114.2.7 `void ibis::zone::write (const char * dt) const` [virtual]

Save index to a file.

Outputs the index in a compact binary format to the named file or directory. The index file contains a header that can be identified by the function `isIndex`.

Reimplemented from [ibis::bin](#).

The documentation for this class was generated from the following files:

- [ibin.h](#)
- `ixzone.cpp`

4 FastBit File Documentation

4.1 array_t.h File Reference

Definition of template [array_t](#).

```
#include "fileManager.h"
#include "horometer.h"
#include <iomanip>
```

Namespaces

- namespace [ibis](#)
- namespace **ibis::util**

Classes

- class [array_t< T >](#)
Template [array_t](#) implements a replacement of `std::vector`.

Functions

- `template<class T> void ibis::util::reorder (array_t< T * > &arr, const array_t< uint32_t > &ind)`
- `template<class T> void ibis::util::reorder (array_t< T > &arr, const array_t< uint32_t > &ind)`

4.1.1 Detailed Description

Definition of template [array_t](#).

Note:

[array_t](#) is not in the name space `ibis` because the compilers used during the early development of this project did not accept templates inside a name space.

4.2 bitvector.h File Reference

Definition of Word-Aligned Hybrid code.

```
#include "array_t.h"
#include <stdio.h>
#include <limits.h>
#include <iomanip>
```

Classes

- class [ibis::bitvector](#)
A data structure to represent a sequence of bits.
- struct **ibis::bitvector::active_word**
The struct active_word stores the last few bits that do not fill a whole word.
- class [ibis::bitvector::const_iterator](#)
The const_iterator class. It iterates on the individual bits.
- class [ibis::bitvector::indexSet](#)
The indexSet stores positions of bits that are one.
- class [ibis::bitvector::iterator](#)
The iterator that allows modification of bits.
- struct **ibis::bitvector::run**
An internal struct used during logical operations to track the usage of fill words.

Functions

- `std::ostream & operator<< (std::ostream &, const ibis::bitvector &)`

4.2.1 Detailed Description

Definition of Word-Aligned Hybrid code.

4.3 bitvector64.h File Reference

Definition of 64-bit version of the Word-Aligned Hybrid code.

```
#include "array_t.h"  
#include <stdio.h>  
#include <limits.h>  
#include <iomanip>
```

Classes

- class [ibis::bitvector64](#)
A data structure to represent a sequence of bits.
- struct **ibis::bitvector64::active_word**
The struct active_word stores the last few bits that do not fill a whole word.
- class [ibis::bitvector64::const_iterator](#)
The const_iterator class. It iterates on the individual bits.
- class [ibis::bitvector64::indexSet](#)
The indexSet stores positions of bits that are one.

- class `ibis::bitvector64::iterator`
The iterator that allows modification of bits.
- struct `ibis::bitvector64::run`
An internal struct used during logical operations to track the usage of fill words.

Functions

- `std::ostream & operator<<` (`std::ostream &`, `const ibis::bitvector64 &`)

4.3.1 Detailed Description

Definition of 64-bit version of the Word-Aligned Hybrid code.

4.4 bord.h File Reference

Defines `ibis::bord`.

```
#include "table.h"
#include "util.h"
#include "part.h"
```

Namespaces

- namespace `ibis`

Classes

- class `ibis::bord`
Class `ibis::bord` stores all its data in memory.
- class `ibis::bord::column`
An in-memory version of `ibis::column`.
- class `ibis::bord::cursor`
- struct `ibis::bord::cursor::bufferElement`
- class `ibis::bord::part`

4.4.1 Detailed Description

Defines `ibis::bord`.

This is an in-memory table, which a single data partition completely residing in memory.

4.5 bundle.h File Reference

Designed to store selected values.

```
#include "util.h"
```

```
#include "array_t.h"
#include "query.h"
#include "column.h"
#include "colValues.h"
```

Namespaces

- namespace [ibis](#)

Classes

- class [ibis::bundle](#)
The public interface of bundles.
- class [ibis::bundle0](#)
The null bundle. It contains only a list of RIDs.
- class [ibis::bundle1](#)
The bundle with only one component.
- class [ibis::bundles](#)
The bundle with multiple components.
- class [ibis::query::result](#)
The class [ibis::query::result](#) allows user to retrieve query result one row at a time.

4.5.1 Detailed Description

Designed to store selected values.

The class [ibis::bundle](#) is use to represent a sorted version of the selected columns of a query. The selected columns can be of any type. The string values are internally recorded as integers. The bundles are written to a directory containing other type of information about the query.

This is an incore implementation, that is, it stores all relevant values in memory. It is intended to be used only to sort the selected values and immediately write out the content to files.

When multiple components are selected, a generic version of the sorting algorithm is used. It should be faster to handle special versions separately. For example, if all the selected components are of the same type, it is possible to use a more compact array structure for comparisons. It might be also useful to separate out the case where there are only two components.

4.6 capi.h File Reference

This header file defines a C API for accessing functions of FastBit IBIS implementations.

Typedefs

- typedef FastBitQuery * [FastBitQueryHandle](#)
A handle to use used by C client.
- typedef FastBitResultSet * **FastBitResultSetHandle**

Functions

- int `fastbit_build_index` (const char *indexLocation, const char *cname, const char *indexOptions)
Build an index for the named attribute.
- int `fastbit_build_indexes` (const char *indexLocation)
Build indices for all attributes in the named directory.
- `FastBitQueryHandle` `fastbit_build_query` (const char *selectClause, const char *indexLocation, const char *queryConditions)
Build a new FastBit query.
- `FastBitResultSetHandle` `fastbit_build_result_set` (`FastBitQueryHandle` query)
Build a new result set from a query object.
- void `fastbit_cleanup` (void)
Clean up resources hold by FastBit file manager.
- int `fastbit_destroy_query` (`FastBitQueryHandle` query)
Free all resource associated with the handle.
- int `fastbit_destroy_result_set` (`FastBitResultSetHandle` rset)
Destroy a result set.
- const char * `fastbit_get_from_clause` (`FastBitQueryHandle` query)
Return the table name.
- const double * `fastbit_get_qualified_double` (`FastBitQueryHandle` query, const char *cname)
- const float * `fastbit_get_qualified_float` (`FastBitQueryHandle` query, const char *cname)
- const int32_t * `fastbit_get_qualified_int` (`FastBitQueryHandle` query, const char *cname)
Return a pointer to an array holding the values of attribute `att` that qualifies the specified selection conditions.
- const uint32_t * `fastbit_get_qualified_unsigned` (`FastBitQueryHandle` query, const char *cname)
- int `fastbit_get_result_columns` (`FastBitQueryHandle` query)
Count the number of columns selected in the select clause of the query.
- int `fastbit_get_result_rows` (`FastBitQueryHandle` query)
Return the number of hits in the query.
- const char * `fastbit_get_select_clause` (`FastBitQueryHandle` query)
Return the string form of the select clause.
- int `fastbit_get_verbose_level` (void)
Return the current verbosity level.
- const char * `fastbit_get_where_clause` (`FastBitQueryHandle` query)
Return the where clause of the query.
- void `fastbit_init` (const char *rcfile)
Initialization function.
- int `fastbit_purge_index` (const char *indexLocation, const char *cname)
Purge the index of the named attribute.

- int `fastbit_purge_indexes` (const char *indexLocation)
Purge all index files.
- double `fastbit_result_set_get_double` (FastBitResultSetHandle rset, const char *cname)
Get the value of the named column as a double-precision floating-point number.
- float `fastbit_result_set_get_float` (FastBitResultSetHandle rset, const char *cname)
Get the value of the named column as a single-precision floating-point number.
- int `fastbit_result_set_get_int` (FastBitResultSetHandle rset, const char *cname)
Get the value of the named column as an integer.
- const char * `fastbit_result_set_get_string` (FastBitResultSetHandle rset, const char *cname)
Get the value of the named column as a string.
- unsigned `fastbit_result_set_get_unsigned` (FastBitResultSetHandle rset, const char *cname)
Get the value of the named column as an unsigned integer.
- double `fastbit_result_set_getDouble` (FastBitResultSetHandle rset, unsigned position)
Get the value of the named column as a double-precision floating-point number.
- float `fastbit_result_set_getFloat` (FastBitResultSetHandle rset, unsigned position)
Get the value of the named column as a single-precision floating-point number.
- int32_t `fastbit_result_set_getInt` (FastBitResultSetHandle rset, unsigned position)
Get the value of the named column as an integer.
- const char * `fastbit_result_set_getString` (FastBitResultSetHandle rset, unsigned position)
Get the value of the named column as a string.
- uint32_t `fastbit_result_set_getUnsigned` (FastBitResultSetHandle rset, unsigned position)
Get the value of the named column as an unsigned integer.
- int `fastbit_result_set_next` (FastBitResultSetHandle rset)
Returns 0 if there are more results, otherwise returns -1.
- int `fastbit_set_verbose_level` (int v)
Change the verbosity of FastBit functions.

4.6.1 Detailed Description

This header file defines a C API for accessing functions of FastBit IBIS implementations.

It deals with data tables as directories and queries as pointers to struct FastBitQuery.

Note:

For functions that return integer error code, 0 always indicate success, a negative number indicate error, a positive number may also be returned to carry results, such as in `fastbit_get_result_size`.

For functions that returns pointers, they may return a nil pointer in case of error.

4.6.2 Function Documentation

4.6.2.1 **FastBitQueryHandle** fastbit_build_query (const char * *selectClause*, const char * *indexLocation*, const char * *queryConditions*)

Build a new FastBit query.

This is logically equivalent to the SQL statement "SELECT selectClause FROM indexLocation WHERE queryConditions." A blank selectClause is equivalent to "count(*)".

Note:

Must call fastbit_destroy_query on the handle returned to free the resources.

4.6.2.2 int fastbit_destroy_query (**FastBitQueryHandle** *query*)

Free all resource associated with the handle.

Note:

The handle becomes invalid.

4.6.2.3 const int32_t* fastbit_get_qualified_int (**FastBitQueryHandle** *query*, const char * *cname*)

Return a pointer to an array holding the values of attribute `att` that qualifies the specified selection conditions.

Note:

The caller can NOT free the memory pointed by the pointer returned. Must call fastbit_destroy_query to free the memory after use. This applies to all other versions of fastbit_get_qualified_ttt.

4.6.2.4 int fastbit_get_result_rows (**FastBitQueryHandle** *query*)

Return the number of hits in the query.

It is also the number of rows in the result set. The arrays returned by fastbit_get_qualified_xxx shall have this many elements.

4.6.2.5 void fastbit_init (const char * *rcfile*)

Initialization function.

May pass in a nil pointer if one is expected to use the default configuration files listed in the documentation of `ibis::resources::read`. One may call this function multiple times to read multiple configuration files to modify the parameters.

4.6.2.6 int32_t fastbit_result_set_getInt (**FastBitResultSetHandle** *rset*, unsigned *position*)

Get the value of the named column as an integer.

The argument `index` is the position (starting with 0) of the attribute in the select clause. This should be faster than the one with `cname` as argument since it avoids name look up.

4.7 category.h File Reference

Define two specialization of the column class.

```
#include "irelic.h"
#include "column.h"
```

Classes

- class [ibis::category](#)
A specialized low-cardinality text field.
- class [ibis::dictionary](#)
Provide a mapping between strings and integers.
- class [ibis::text](#)
A minimalistic structure for storing arbitrary text fields.

4.7.1 Detailed Description

Define two specialization of the column class.

IBIS represents incoming data table with vertical partitioning. Each column object represents one column of the relational table. The terms used to describe each column of the table are strongly influenced by the first project using this software, a high-energy physics experiment named STAR.

4.8 column.h File Reference

Define the class column.

```
#include "table.h"  
#include "qExpr.h"  
#include "bitvector.h"  
#include <string>
```

Namespaces

- namespace [ibis](#)

Classes

- class [ibis::column](#)
The class to represent a column of a data table.
- class [ibis::column::indexLock](#)
A class for controlling access of the index object of a column.
- class [ibis::column::info](#)
Some basic information about a column.
- class [ibis::column::mutexLock](#)
Provide a mutual exclusion lock on an [ibis::column](#).
- class [ibis::column::writeLock](#)
Provide a write lock on a [ibis::column](#) object.

Functions

- `std::ostream & operator<<` (`std::ostream &out`, `const ibis::column &prop`)

4.8.1 Detailed Description

Define the class `column`.

A column of a relational table is also known as an attribute of a relation. In IBIS, columns are stored separate from each other. This storage strategy is commonly known as vertical partitioning.

4.9 colValues.h File Reference

A set of utility classes for storing the selected values.

```
#include "column.h"
```

Classes

- class `ibis::colDoubles`
A class to store double precision floating-point values.
- class `ibis::colFloats`
A class to store single precision float-point values.
- class `ibis::colInts`
A class to store integer values.
- class `ibis::colLongs`
A class to store integer values.
- class `ibis::colUInts`
A class to store unsigned integer values.
- class `ibis::colULongs`
A class to store unsigned integer values.
- class `ibis::colValues`
A pure virtual base class.

4.9.1 Detailed Description

A set of utility classes for storing the selected values.

4.10 const.h File Reference

Defines common data types, constants and macros.

```
#include <errno.h>  
#include <pthread.h>  
#include <sys/types.h>
```

```
#include <string.h>
#include <functional>
#include <iostream>
#include <stdint.h>
#include <syslimits.h>
#include <strings.h>
```

Namespaces

- namespace [ibis](#)

Classes

- struct [_rwlock](#)
- struct [ibis::lessi](#)
A case-insensitive version of less for comparing names of tables, columns, and resources.
- union [ibis::rid_t](#)
The object identifiers used to distinguish records.
- struct [ibis::rid_t::name](#)
As two 32-bit values.

Defines

- #define **DIRSEP** '/'
- #define **IBIS_REPLACEMENT_RWLOCK**
- #define **int16_t** short int
- #define **int32_t** int
- #define **int64_t** long long int
- #define **MAX_LINE** 2048
- #define **MessageBox**(x1, x2, x3, x4) ; { }
- #define **PATH_MAX** 512
- #define [PREFERRED_BLOCK_SIZE](#) 1048576
PREFERRED_BLOCK_SIZE is the parameter used to determine the logical page size during some I/O intensive operations, such as nested loop join.
- #define **REASON** " " << strerror(errno) << std::endl;
- #define **stricmp** strcasecmp
- #define **strnicmp** strncasecmp
- #define **THREAD_RWLOCK_INITIALIZER**
- #define **TIME_BUF_LEN** 32
- #define **uint16_t** unsigned short int
- #define **uint32_t** unsigned int
- #define **uint64_t** unsigned long long int

Typedefs

- typedef [_rwlock](#) **pthread_rwlock_t**

Functions

- int **pthread_rwlock_destroy** (pthread_rwlock_t *rwlock)
- int **pthread_rwlock_init** (pthread_rwlock_t *rwlock, void *)
- int **pthread_rwlock_rdlock** (pthread_rwlock_t *rwlock)
- int **pthread_rwlock_tryrdlock** (pthread_rwlock_t *rwlock)
- int **pthread_rwlock_trywrlock** (pthread_rwlock_t *rwlock)
- int **pthread_rwlock_unlock** (pthread_rwlock_t *rwlock)
- int **pthread_rwlock_wrlock** (pthread_rwlock_t *rwlock)

Variables

- int **ibis::gVerbose**
Verbosity level.

4.10.1 Detailed Description

Defines common data types, constants and macros.

Used by all files in the IBIS implementation of FastBit from the Scientific Data Management Research Group of Lawrence Berkeley National Laboratory.

4.10.2 Define Documentation

4.10.2.1 #define PREFERRED_BLOCK_SIZE 1048576

PREFERRED_BLOCK_SIZE is the parameter used to determine the logical page size during some I/O intensive operations, such as nested loop join.

Many CPUs have 512KB cache, setting this value to 256K (262144) will allow about two such 'logical' block to be in cache at the same time, which should be good to things like nested loop join.

4.10.2.2 #define THREAD_RWLOCK_INITIALIZER

Value:

```
{PTHREAD_MUTEX_INITIALIZER, PTHREAD_COND_INITIALIZER, \
    PTHREAD_COND_INITIALIZER, 0, 0}
```

4.11 fileManager.h File Reference

Defines a simple file Manager.

```
#include "util.h"
#include <set>
#include <map>
#include <math.h>
```

Classes

- class **ibis::fileManager**
This fileManager is intended to allow different objects to share the same open file.

- class [ibis::fileManager::cleaner](#)
A function object to be used to register external cleaners.
- class **ibis::fileManager::mutexLock**
Used to prevent simultaneous modification of the two internal lists.
- class [ibis::fileManager::readLock](#)
An object who uses a file under the management of the file manager should hold a [readLock](#).
- class [ibis::fileManager::roFile](#)
This class manages content of a whole (read-only) file.
- class [ibis::fileManager::storage](#)
The storage class treats all memory as char.*
- class **ibis::fileManager::writeLock**
A write lock for controlling access to the two interval lists.

Defines

- #define **HAVE_MMAP** defined(unix)||defined(linux)||defined(__APPLE__)||defined(__CYGWIN__)

4.11.1 Detailed Description

Defines a simple file Manager.

Note:

Use malloc and realloc to manage memory when the file content is actually in memory. The main reason for doing so is to malloc for resizing. This may potentially cause problems with memory allocation through the new operator provided by C++ compiler.

4.12 horometer.h File Reference

Defines a simple timer class.

```
#include <stdio.h>
#include <time.h>
```

Namespaces

- namespace [ibis](#)

Classes

- class [ibis::horometer](#)
Horometer – a primitive timing instrument.

4.12.1 Detailed Description

Defines a simple timer class.

4.13 `ibin.h` File Reference

Define `ibis::bin` and derived classes.

```
#include "index.h"
```

Classes

- class `ibis::ambit`
The multi-level range based (cumulative) index.
- class `ibis::bak`
Maps each value to a lower prevision (decimal) values and use the the low precision value as center of the bin.
- class `ibis::bak2`
A variation on `ibis::bak`, it splits each bin of `ibis::bak` in two, one for entries less than the mapped value and one for the entries that greater and equal to the mapped value.
- struct `ibis::bak2::grain`
A simple structure to record the position of the values mapped to the same value.
- struct `ibis::bak::grain`
- class `ibis::bin`
The equality encoded bitmap index with binning.
- struct `ibis::bin::granule`
A data structure to assist the mapping of values to lower precisions.
- class `ibis::egale`
The multicomponent equality code on bins.
- class `ibis::entre`
The multicomponent interval code on bins.
- class `ibis::mesa`
This class implements the two-side range encoding from Chan and Ioannidis.
- class `ibis::moins`
The multicomponent range code on bins.
- class `ibis::pack`
A two-level index.
- class `ibis::pale`
A two-level index.
- class `ibis::range`
The range encoded bitmap index based.
- class `ibis::zone`
A two-level index.

4.13.1 Detailed Description

Define `ibis::bin` and derived classes.

```
bin -> range, mesa, ambit, pale, pack, zone, egale, bak, bak2
egale -> moins, entre
```

4.14 **ibis.h File Reference**

The header file to be included by all user code.

```
#include "meshQuery.h"
#include "resource.h"
#include "bundle.h"
#include "query.h"
#include "part.h"
#include "rids.h"
```

Namespaces

- namespace `ibis`

Functions

- void `ibis::init` (const int verbose=0, const char *rcfile=0)
Initializes internal resources required by ibis.

4.14.1 Detailed Description

The header file to be included by all user code.

It defines all classes and functions intended to use `ibis::part` interface. All such classes and functions are defined in the namespace `ibis`. Before performing any operations, the first function to be called is `ibis::init`.

See also:

`ibis::init`

4.15 **idirekte.h File Reference**

This is an implementation of the the simple bitmap index without the first binning step.

```
#include "index.h"
```

Classes

- class `ibis::direkte`
Directly use the integer values as bin number to avoid some intermediate steps.

4.15.1 Detailed Description

This is an implementation of the the simple bitmap index without the first binning step.

It directly uses the integer values as bin number. The word `direkte` in Danish means `direct`.

4.16 `ibis::keywords.h` File Reference

This index is a keyword index for a string-valued column.

```
#include "index.h"
#include "category.h"
```

Classes

- class `ibis::keywords`
Class `ibis::keywords` defines a boolean term-document matrix.

4.16.1 Detailed Description

This index is a keyword index for a string-valued column.

It contains a boolean version of the term-document matrix and supports exact matches of keywords/terms.

4.17 `index.h` File Reference

Definition of the common functions of an index.

```
#include "qExpr.h"
#include "bitvector.h"
#include <string>
```

Namespaces

- namespace `ibis`

Classes

- class `ibis::index`
The base index class.
- class `ibis::index::barrel`
A specialization that adds function `setValue`.

4.17.1 Detailed Description

Definition of the common functions of an index.

The index class is a pure virtual base class with a static `create` function and a few virtual functions that provide common functionality.

An index is built for each individual column (`ibis::column`) of a data table. The primary function of the index is to compute the solution or an estimation (as a pair of upper and lower bounds) for a range query. It needs to be generated and updated as necessary. The simplest way of generating an index is to build one from a file containing the binary values of a column. An index can only be updated for new records appended to the data table. Any other form of update, such as removal of some records, change some existing records can only be updated by removing the existing index then recreate the index.

4.18 irelic.h File Reference

Define `ibis::relic` and its derived classes.

```
#include "index.h"
```

Classes

- class `ibis::bylt`
The two-level range-equality code.
- class `ibis::fade`
The multicomponent range-encoded index.
- class `ibis::fuzz`
The two-level interval-equality code.
- class `ibis::relic`
The basic bitmap index.
- class `ibis::sapid`
The multicomponent equality encoded index.
- class `ibis::sbiad`
The multicomponent interval encoded index.
- class `ibis::slice`
The bit-sliced index (O'Neil). It used the binary encoding.
- class `ibis::zona`
The two-level equality-equality code.

4.18.1 Detailed Description

Define `ibis::relic` and its derived classes.

```
relic -> slice, fade, bylt(pack), zona (zone)
fade -> sbiad, sapid
```

4.19 iroster.h File Reference

Defines a pseudo-index.

```
#include "array_t.h"
#include "util.h"
```

Classes

- class [ibis::roster](#)

A roster list is a list of indices for ordering the values in the ascending order.

4.19.1 Detailed Description

Defines a pseudo-index.

Used in some performance comparisons.

4.20 mensa.h File Reference

A table with multiple data partitions on disk.

```
#include "table.h"
#include "util.h"
#include "fileManager.h"
```

Namespaces

- namespace [ibis](#)

Classes

- class [ibis::mensa](#)

Class [ibis::mensa](#) contains multiple (horizontal) data partitions ([ibis::part](#)) to form a logical data table.

- class [ibis::mensa::cursor](#)
- struct [ibis::mensa::cursor::bufferElement](#)

4.20.1 Detailed Description

A table with multiple data partitions on disk.

This class defines the data structure to encapsulate multiple on-disk data partitions into a logical table. The class translates the function defined on [ibis::part](#) to the [ibis::table](#) interface.

4.21 meshQuery.h File Reference

The header file defining an extension of query on mesh data.

```
#include "query.h"
```

Namespaces

- namespace [ibis](#)

Classes

- class [ibis::meshQuery](#)
The class adds more functionality to [ibis::query](#) to handle data from meshes.

4.21.1 Detailed Description

The header file defining an extension of query on mesh data.

4.22 part.h File Reference

Define the class [ibis::part](#).

```
#include "column.h"
#include "resource.h"
#include <string>
#include <vector>
#include <functional>
```

Namespaces

- namespace [ibis](#)
- namespace [ibis::util](#)

Classes

- class [ibis::part](#)
The class [ibis::part](#) represents a partition of a relational table.
- class [ibis::part::advisoryLock](#)
An non-blocking version of [writeLock](#).
- class [ibis::part::barrel](#)
To read a list of variables at the same time.
- class [ibis::part::cleaner](#)
A cleaner to be used by the [fileManager::unload](#) function.
- struct [ibis::part::indexBuilderPool](#)
- struct [ibis::part::info](#)
A simple class to describe an [ibis::part](#) object.
- class [ibis::part::mutexLock](#)
Provide a mutual exclusion lock on an [ibis::part](#) object.
- class [ibis::part::readLock](#)
Provide a read lock on an [ibis::part](#).
- struct [ibis::part::thrArg](#)
- class [ibis::part::vault](#)

To read variables in certain order.

- class [ibis::part::writeLock](#)
Provide a write lock on an [ibis::part](#).

Functions

- void [ibis::util::tablesFromDir](#) (ibis::partList &tlist, const char *dir1)
Look into the given directory for table.tdc files.
- void [ibis::util::tablesFromDir](#) (ibis::partList &tables, const char *adir, const char *bdir)
Look for data directories in the given pair of directories.
- void [ibis::util::tablesFromResources](#) (ibis::partList &tables, const [ibis::resource](#) &res)
Reconstruct partitions using data directories specified in the resources.

4.22.1 Detailed Description

Define the class [ibis::part](#).

This class defines some rudimentary functions for managing a vertically partitioned data partition and answering simple queries. It also provides limited number of functions to modify the data partition.

4.23 **predicate.h** File Reference

Replaces some default lex functions.

```
#include "qExpr.h"  
#include <stdio.h>  
#include <stack>  
#include <deque>  
#include <vector>
```

Namespaces

- namespace [ibis](#)

Defines

- #define **YYLMAX** BUFSIZ

Functions

- int **lex_input** ()
- qExpr * [ibis::parseQuery](#) (const char *str)
Parse a query string.
- void **unput** (int)
- void **yyerror** (const char *s)

- int `yylex ()`
- int `yyparse ()`
- void `yyparse_cleanup ()`
- int `yywrap ()`

Variables

- int `parse_length`
- int `parse_offset`
- `std::vector< char * >` `parse_str_vec`
- `char *` `parse_string`
- `char` `yytext` [YYLMAX]

4.23.1 Detailed Description

Replaces some default lex functions.

The replacement functions all it to work on a string named `parse_string` (size `parse_length`). ALL lex/yacc related variables are in global namespace, only `parseQuery` is in the usual `ibis` namespace.

Note:

The only function that should be used elsewhere is `parseQuery`

```
ibis::qExpr* ibis::parseQuery(const char* str)
```

4.24 qExpr.h File Reference

Define the query expression.

```
#include "util.h"
#include <functional>
```

Namespaces

- namespace `ibis`

Classes

- class `ibis::compRange`
The class `compRange` stores computed ranges.
- class `ibis::compRange::barrel`
A barrel to hold a list of variables.
- class `ibis::compRange::bediener`
- class `ibis::compRange::literal`
- class `ibis::compRange::number`
- class `ibis::compRange::stdFunction1`
- class `ibis::compRange::stdFunction2`
- class `ibis::compRange::term`
- class `ibis::compRange::variable`
- class `ibis::qAnyAny`

A user specifies this type of query expression with the following syntax,.

- class `ibis::qContinuousRange`
Simple range condition.
- class `ibis::qDiscreteRange`
- class `ibis::qExpr`
The top level query expression object.
- struct `ibis::qExpr::weight`
A functor to be used by the function reorder.
- class `ibis::qMultiString`
- class `ibis::qRange`
A class to represent simple range conditions.
- class `ibis::qString`
The class `qString` encapsulates information for comparing string values.
- class `ibis::rangeJoin`
A join is defined by two names and a numerical expression.

Functions

- `std::ostream & operator<<` (`std::ostream &out`, `const ibis::qExpr &pn`)
Wrap function print as `operator<<`.

4.24.1 Detailed Description

Define the query expression.

4.25 query.h File Reference

The header file defining the individual query objects.

```
#include "part.h"  
#include <map>
```

Namespaces

- namespace `ibis`

Classes

- class `ibis::query`
A data structure for representing user queries.
- class `ibis::query::readLock`
- class `ibis::query::weight`
- class `ibis::query::writeLock`

4.25.1 Detailed Description

The header file defining the individual query objects.

4.26 resource.h File Reference

Defines a class to hold name-value pairs.

```
#include "util.h"
#include <fstream>
#include <map>
```

Namespaces

- namespace [ibis](#)

Classes

- class [ibis::resource](#)
A container for name-value pairs.

Functions

- [ibis::resource & ibis::gParameters \(\)](#)
The reference to the global configuration parameters.

4.26.1 Detailed Description

Defines a class to hold name-value pairs.

4.27 rids.h File Reference

Define simple IO functions for [ibis::rid_t](#).

```
#include "util.h"
```

Namespaces

- namespace [ibis](#)

Classes

- class [ibis::ridHandler](#)
A class for handling file IO for [ibis::rid_t](#).

4.27.1 Detailed Description

Define simple IO functions for [ibis::rid_t](#).

Based on on [OidIOHandler](#) by David Malon <malon@anl.gov>.

4.28 tab.h File Reference

This file stores two trivial concrete classes of [ibis::table](#): `tabula` and `tabele`.

```
#include "table.h"
#include <iostream>
```

Namespaces

- namespace [ibis](#)

Classes

- class [ibis::tabele](#)
A trivial class for table with one row and one column.
- class [ibis::tabele::cursor](#)
- class [ibis::tabula](#)
A trivial class for table with no columns.
- class [ibis::tabula::cursor](#)

4.28.1 Detailed Description

This file stores two trivial concrete classes of [ibis::table](#): `tabula` and `tabele`.

Here is an explanation of how these two words are related to "table".

Remarks:

The term "table" is derived from a merger of French `table` and Old English `tabele`, ultimately from the Latin word `tabula`, "a board, plank, flat piece". In Late Latin, `tabula` took over the meaning previously reserved to `mensa` (preserved in Spanish `mesa` "table"). In Old English, the word replaced `bord` for this meaning. – Wikipedia.

4.29 table.h File Reference

FastBit Table Interface.

```
#include <ostream>
#include <vector>
#include <map>
#include <string>
#include <cstdlib>
#include "const.h"
```

Namespaces

- namespace [ibis](#)

Classes

- class [ibis::table](#)
The abstract table class.
- class [ibis::table::cursor](#)
Cursor class for row-wise data accesses.
- struct [ibis::table::row](#)
A simple struct for storing a row of a table.
- class [ibis::tableList](#)
A list of tables.
- class [ibis::tablex](#)
The class for expandable tables.

Enumerations

- enum [ibis::TYPE_T](#) {
[ibis::UNKNOWN_TYPE](#) = 0, [ibis::OID](#), [ibis::BYTE](#), [ibis::UBYTE](#),
[ibis::SHORT](#), [ibis::USHORT](#), [ibis::INT](#), [ibis::UINT](#),
[ibis::LONG](#), [ibis::ULONG](#), [ibis::FLOAT](#), [ibis::DOUBLE](#),
[ibis::CATEGORY](#), [ibis::TEXT](#) }
Supported data types.

Variables

- const char * [ibis::TYPECODE](#)
One-character code for the enumeration types.
- const char ** [ibis::TYPESTRING](#)
Human readable version of the enumeration types.

4.29.1 Detailed Description

FastBit Table Interface.

This is intended to be a facade for FastBit functions that provide a view that is all operations are on tables. Two main classes are defined here, `table` and `tablex`. The class `table` is intended to be used with read-only data and it provides mostly querying functions. The class `tablex` is intended for users to add new records to the existing records.

4.30 `tafel.h` File Reference

An expandable table.

```
#include "table.h"
#include "bitvector.h"
```

Namespaces

- namespace [ibis](#)

Classes

- class [ibis::tafel](#)
An expandable table.
- struct [ibis::tafel::column](#)

4.30.1 Detailed Description

An expandable table.

This file defines [ibis::tafel](#).

4.31 `twister.h` File Reference

Pseudorandom number generators.

```
#include <time.h>
#include <math.h>
#include <limits.h>
#include <float.h>
#include <iostream>
#include <vector>
```

Namespaces

- namespace [ibis](#)

Classes

- class [ibis::discretePoisson](#)
Discrete random number with Poisson distribution $\exp(-x/\lambda)$.
- class [ibis::discretePoisson1](#)
Specialized version of the Poisson distribution $\exp(-x)$.
- class [ibis::discreteZipf](#)
Discrete Zipf distribution: $p(k)$ is proportional to $(v+k)^{-a}$ where $a > 1$, $k \geq 0$.
- class [ibis::discreteZipf1](#)

A specialized case of the Zipf distribution $f(x) = 1/(1+x)$.

- class [ibis::discreteZipf2](#)
A specialized version of the Zipf distribution $f(x) = 1/(1+x)^2$.
- class [ibis::MersenneTwister](#)
Mersenne Twister generates uniform random numbers efficiently.
- class [ibis::uniformRandomNumber](#)
A functor to generate uniform random number in the range $[0, 1)$.

4.31.1 Detailed Description

Pseudorandom number generators.

MersenneTwister: A C++ class that use the similar interface as `java.util.Random`. The basic algorithm is based on the Mersenne Twister by M. Matsumoto and T. Nishimura.

MersenneTwister also include a function called `nextZipf` to generate Zipf distributed random numbers (floats).

This file also contains additional classes that generate discrete Zipf and Poisson distributions (named `discreteZipf` and `discretePoisson`)

4.32 util.h File Reference

Defines minor utility functions and common classes used by FastBit.

```
#include "const.h"
#include <ctype.h>
#include <stdio.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <map>
#include <vector>
#include <string>
#include <float.h>
#include <math.h>
#include <unistd.h>
```

Namespaces

- namespace [ibis](#)
- namespace [ibis::util](#)
- namespace [std](#)

Classes

- class [ibis::bad_alloc](#)
A specialization of `std::bad_alloc`.

- class `ibis::nameList`
A data structure to store a small set of names.
- class `ibis::selected`
A data structure to store the select clause of a query.
- class `ibis::util::counter`
A simple global counter.
- class `ibis::util::ioLock`
A global IO mutex lock.
- class `ibis::util::mutexLock`
An wrapper class for perform `pthread_mutex_lock/unlock`.
- class `ibis::util::quietLock`
An wrapper class for perform `pthread_mutex_lock/unlock`.
- class `ibis::util::readLock`
An wrapper class for perform `pthread_rwlock_rdlock/unlock`.
- class `ibis::util::writeLock`
An wrapper class for perform `pthread_rwlock_wrlock/unlock`.
- struct `std::less< char * >`
- struct `std::less< const char * >`
- struct `std::less< const ibis::rid_t * >`
- struct `std::less< ibis::rid_t >`

Defines

- `#define DBL_EPSILON 2.2204460492503131e-16`
- `#define OPEN_FILEMODE S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP | S_IROTH`
- `#define Stat_T struct stat`
- `#define UnixClose ::close`
- `#define UnixFStat ::fstat`
- `#define UnixOpen ::open`
- `#define UnixRead ::read`
- `#define UnixSeek ::lseek`
- `#define UnixSnprintf ::snprintf`
- `#define UnixStat ::stat`
- `#define UnixWrite ::write`

Typedefs

- `typedef std::vector< colValues * > ibis::colList`
- `typedef std::map< const char *, part *, lessi > ibis::partList`
- `typedef array_t< rid_t > ibis::RIDSet`

Functions

- `uint32_t ibis::util::checksum` (`uint32_t a`, `uint32_t b`)
Fletcher's checksum on two integers. Returns an integer.
- `double ibis::util::coarsen` (`const double in`, `const unsigned prec`)
Reduce the decimal precision of the incoming floating-point value to specified precision.
- `int ibis::util::copy` (`const char *to`, `const char *from`)
Copy "from" to "to".
- `uint32_t ibis::util::getBuffer` (`char *&buf`)
Generate a reasonably sized buffer for storing temporary contents.
- `off_t ibis::util::getFileSize` (`const char *name`)
Return size of the file in bytes.
- `void ibis::util::getGMTime` (`char *str`)
- `void ibis::util::getLocalTime` (`char *str`)
Return the current time in string format as `asctime_r`.
- `void ibis::util::getString` (`std::string &str`, `const char *&buf`, `const char *delim`)
Extract the next quoted string or the blank delimited string.
- `char * ibis::util::getString` (`const char *buf`)
Treat all bytes in `buf` as the string.
- `const char * ibis::util::getToken` (`char *&str`, `const char *tok_chrs`)
- `char * ibis::util::itoa` (`int value`, `char *str`, `int`)
- `void ibis::util::logMessage` (`const char *event`, `const char *fmt`,...)
Print a message to standard output.
- `int ibis::util::makeDir` (`const char *dir`)
Recursively create the name directory.
- `std::ostream & operator<<` (`std::ostream &out`, `const ibis::rid_t &rid`)
Print a `rid_t` to an output stream.
- `std::istream & operator>>` (`std::istream &is`, `ibis::rid_t &rid`)
Read a `rid_t` from an input stream.
- `const ibis::bitvector64 & ibis::outerProduct` (`const ibis::bitvector &a`, `const ibis::bitvector &b`, `ibis::bitvector64 &c`)
Compute the outer product of `a` and `b`, add the result to `c`.
- `const ibis::bitvector64 & ibis::outerProductUpper` (`const ibis::bitvector &a`, `const ibis::bitvector &b`, `ibis::bitvector64 &c`)
Add the strict upper triangular portion of the outer production between `a` and `b` to `c`.
- `double ibis::util::rand` ()
A pseudo-random number generator (0,1).
- `double ibis::util::rand` ()

A pseudo-random number generator (0,1).

- int **ibis::util::readDouble** (double &val, const char *&str, const char *del)
Attempt to convert the incoming string into a double.
- int **ibis::util::readInt** (int64_t &val, const char *&str, const char *del)
Attempt to convert the incoming string into an integer.
- void **ibis::util::removeDir** (const char *name, bool leaveDir=false)
Remove the content of named directory.
- void **ibis::util::removeTail** (char *str, char tail)
Remove trailing character 'tail' from str.
- void **ibis::util::removeTail** (char *str, char tail)
Remove trailing character 'tail' from str.
- void **ibis::util::secondsToString** (const time_t, char *str)
- std::string **ibis::util::shortName** (const std::string &de)
- bool **ibis::util::strMatch** (const char *str, const char *pat)
Match the string.
- char * **ibis::util::strnewdup** (const char *s, const uint32_t n)
- char * **ibis::util::strnewdup** (const char *s)
duplicate string content with C++ default new operator
- char * **ibis::util::strnewdup** (const char *s, const uint32_t n)
- char * **ibis::util::strnewdup** (const char *s)
duplicate string content with C++ default new operator
- char * **ibis::util::trim** (char *str)
Remove leading and trailing blank space.
- char * **ibis::util::trim** (char *str)
Remove leading and trailing blank space.
- void **ibis::util::uniformFraction** (const long unsigned idx, long unsigned &denominator, long unsigned &numerator)
Compute a denominator and numerator pair to compute a uniform distribution of numbers in a given range.
- unsigned long **ibis::util::uniqueNumber** ()
Return an integer that is always increasing.
- const char * **ibis::util::userName** ()
Return the name of the user who is running the program.
- uint32_t **ibis::util::checksum** (uint32_t a, uint32_t b)
Fletcher's checksum on two integers. Returns an integer.
- uint32_t **ibis::util::checksum** (const char *str, uint32_t sz)
Fletcher's arithmetic checksum with 32-bit result.
- std::string **ibis::util::shortName** (const std::string &longname)

Fletcher's checksum on two integers. Returns an integer.

- double **ibis::util::coarsen** (const double in, const unsigned prec=2)
Reduce the decimal precision of the incoming floating-point value to specified precision.
- double **ibis::util::compactValue** (double left, double right, double start=0.0)
Compute a compact 64-bit floating-point value with a short decimal representation in the range [left, right].
- double **ibis::util::decrDouble** (const double &in)
Reduce the decimal precision of the incoming floating-point value to specified precision.
- void **ibis::util::eq2range** (const double &in, double &left, double &right)
Reduce the decimal precision of the incoming floating-point value to specified precision.
- double **ibis::util::incrDouble** (const double &in)
Functions to handle manipulation of floating-point numbers.
- void **ibis::util::setNaN** (float &val)
Reduce the decimal precision of the incoming floating-point value to specified precision.
- void **ibis::util::setNaN** (double &val)
Set a double to NaN.

- void **ibis::util::int2string** (std::string &str, const std::vector< unsigned > &val)
- void **ibis::util::int2string** (std::string &str, unsigned v1, unsigned v2, unsigned v3)
- void **ibis::util::int2string** (std::string &str, unsigned v1, unsigned v2)
- void **ibis::util::int2string** (std::string &str, unsigned val)
convert 32-bit integers to base-64 printable characters

- void **ibis::util::isortRIDs** (ibis::RIDSet &, uint32_t, uint32_t)
- void **ibis::util::sortRIDs** (ibis::RIDSet &, uint32_t, uint32_t)
- void **ibis::util::sortRIDs** (ibis::RIDSet &rids)
Sorting RID lists.

Variables

- const short unsigned **ibis::util::charIndex** []
- const char * **ibis::util::charTable**
charTable lists the 64 printable characters to be used for names charIndex maps the characters (ASCII) back to integer [0-64]

- pthread_mutex_t **ibis::util::envLock**
A mutex intended to be used for ensuring there is only one function that modifies the environment and other conditions.

4.32.1 Detailed Description

Defines minor utility functions and common classes used by FastBit.

Index

ACCESS_PREFERENCE
 ibis::fileManager, 112
actualMinMax
 ibis::column, 85
add
 ibis::resource, 206
 ibis::tableList, 236
addBins
 ibis::index, 125
addBits
 ibis::index, 125
adjustSize
 ibis::bitvector, 40
 ibis::bitvector64, 45
append
 ibis::column, 85
 ibis::part, 165
 ibis::relic, 202
 ibis::ridHandler, 209
 ibis::tablex, 238
 ibis::tafel, 248
 ibis::text, 250
appendRow
 ibis::tablex, 238
 ibis::tafel, 248
appendRows
 ibis::tablex, 239
 ibis::tafel, 248
appendToBackup
 ibis::part, 165
array_t, 14
 array_t, 17
 bottomk, 17
 deepCopy, 17
 find, 18
 insert, 18
 nosharing, 18
 operator[], 18
 reserve, 18
 size, 18
 stableSort, 18, 19
 topk, 19
array_t.h, 259

binning
 ibis::bin, 32
bitvector
 ibis::bitvector, 40
bitvector.h, 259
bitvector64.h, 260
bitvectorToCoordinates
 ibis::meshQuery, 145
bord.h, 261
bottomk
 array_t, 17
buildIndex
 ibis::bord, 53
 ibis::mensa, 138
 ibis::tabele, 220
 ibis::table, 228
 ibis::tabula, 242
buildIndexes
 ibis::bord, 53
 ibis::mensa, 138
 ibis::tabele, 221
 ibis::table, 228
 ibis::tabula, 242
bundle.h, 261
bylt
 ibis::bylt, 66
BYTE
 ibis, 9

capi.h, 262
 fastbit_build_query, 265
 fastbit_destroy_query, 265
 fastbit_get_qualified_int, 265
 fastbit_get_result_rows, 265
 fastbit_init, 265
 fastbit_result_set_getInt, 265
CATEGORY
 ibis, 9
category.h, 265
checkBin
 ibis::bin, 32
clusteringFactor
 ibis::bitvector, 40
 ibis::bitvector64, 45
column
 ibis::column, 85
column.h, 266
colValues.h, 267
commit
 ibis::part, 165
computeMinMax
 ibis::column, 85, 86
const.h, 267
 PREFERRED_BLOCK_SIZE, 269
 THREAD_RWLOCK_INITIALIZER, 269
construct
 ibis::relic, 202
contractQuery
 ibis::query, 189
countDeltaPairs
 ibis::query, 189
countEqualPairs
 ibis::query, 189
countHits

- ibis::query, 190
- create
 - ibis::index, 125
 - ibis::table, 228
- dataFileName
 - ibis::column, 86
- deepCopy
 - array_t, 17
- divideCounts
 - ibis::index, 126
- doCompare
 - ibis::part, 165, 166
- doScan
 - ibis::part, 166
- DOUBLE
 - ibis, 9
- dump
 - ibis::bord, 53
 - ibis::mensa, 138
 - ibis::tabele, 221
 - ibis::table, 228
 - ibis::tabula, 242
- end
 - ibis::tableList, 236
- estimate
 - ibis::ambit, 21
 - ibis::bin, 32, 33
 - ibis::bord, 53
 - ibis::direkte, 98
 - ibis::egale, 104
 - ibis::entre, 107
 - ibis::index, 126, 127
 - ibis::keywords, 133
 - ibis::mensa, 138
 - ibis::mesa, 143
 - ibis::moins, 148
 - ibis::pack, 151
 - ibis::pale, 154
 - ibis::query, 190
 - ibis::range, 197
 - ibis::relic, 202, 203
 - ibis::slice, 217
 - ibis::tabele, 221
 - ibis::table, 228
 - ibis::tabula, 242
 - ibis::zone, 257
- estimateMatchAny
 - ibis::part, 166
- estimateRange
 - ibis::column, 86
- evaluate
 - ibis::ambit, 21
 - ibis::bin, 33
 - ibis::bylt, 66
 - ibis::direkte, 99
 - ibis::egale, 104
 - ibis::entre, 107
 - ibis::fade, 109
 - ibis::fuzz, 118
 - ibis::index, 127
 - ibis::keywords, 133
 - ibis::mesa, 143
 - ibis::moins, 148
 - ibis::pack, 151
 - ibis::pale, 154
 - ibis::query, 190
 - ibis::range, 197
 - ibis::relic, 203
 - ibis::sapid, 212
 - ibis::sbiad, 213
 - ibis::slice, 217
 - ibis::zona, 255
 - ibis::zone, 258
- evaluateJoin
 - ibis::part, 166
- evaluateRange
 - ibis::column, 86
- expandQuery
 - ibis::query, 190
- expandRange
 - ibis::bak, 24
 - ibis::bak2, 26
 - ibis::bin, 33
 - ibis::index, 127
 - ibis::range, 197
- fastbit_build_query
 - capi.h, 265
- fastbit_destroy_query
 - capi.h, 265
- fastbit_get_qualified_int
 - capi.h, 265
- fastbit_get_result_rows
 - capi.h, 265
- fastbit_init
 - capi.h, 265
- fastbit_result_set_getInt
 - capi.h, 265
- fetch
 - ibis::table::cursor, 233
- fileManager.h, 269
- find
 - array_t, 18
 - ibis::dictionary, 96
 - ibis::nameList, 149
 - ibis::selected, 215
- findString
 - ibis::column, 86
 - ibis::text, 250
- flip
 - ibis::bitvector, 40
 - ibis::bitvector64, 45

- FLOAT
 - [ibis](#), [9](#)
- [fuzz](#)
 - [ibis::fuzz](#), [118](#)
- [getIDDistribution](#)
 - [ibis::part](#), [167](#)
- [getActualMax](#)
 - [ibis::column](#), [86](#)
- [getActualMin](#)
 - [ibis::column](#), [86](#)
- [getColumn](#)
 - [ibis::part](#), [167](#)
- [getColumnAsByte](#)
 - [ibis::table::cursor](#), [234](#)
- [getColumnAsBytes](#)
 - [ibis::bord](#), [53](#)
 - [ibis::mensa](#), [138](#)
 - [ibis::tabela](#), [221](#)
 - [ibis::table](#), [228](#)
 - [ibis::tabula](#), [242](#)
- [getColumnAsDoubles](#)
 - [ibis::bord](#), [53](#)
 - [ibis::mensa](#), [138](#)
 - [ibis::tabela](#), [221](#)
 - [ibis::table](#), [229](#)
 - [ibis::tabula](#), [243](#)
- [getColumnAsFloats](#)
 - [ibis::bord](#), [53](#)
 - [ibis::tabela](#), [221](#)
 - [ibis::table](#), [229](#)
 - [ibis::tabula](#), [243](#)
- [getColumnAsInts](#)
 - [ibis::bord](#), [54](#)
 - [ibis::mensa](#), [138](#)
 - [ibis::tabela](#), [221](#)
 - [ibis::table](#), [229](#)
 - [ibis::tabula](#), [243](#)
- [getColumnAsLongs](#)
 - [ibis::bord](#), [54](#)
 - [ibis::mensa](#), [139](#)
 - [ibis::tabela](#), [222](#)
 - [ibis::table](#), [229](#)
 - [ibis::tabula](#), [243](#)
- [getColumnAsShorts](#)
 - [ibis::bord](#), [54](#)
 - [ibis::mensa](#), [139](#)
 - [ibis::tabela](#), [222](#)
 - [ibis::table](#), [229](#)
 - [ibis::tabula](#), [243](#)
- [getColumnAsStrings](#)
 - [ibis::bord](#), [54](#)
 - [ibis::tabela](#), [222](#)
 - [ibis::table](#), [230](#)
 - [ibis::tabula](#), [243](#)
- [getColumnAsUBytes](#)
 - [ibis::bord](#), [54](#)
 - [ibis::mensa](#), [139](#)
 - [ibis::tabela](#), [222](#)
 - [ibis::table](#), [230](#)
 - [ibis::tabula](#), [244](#)
- [getColumnAsUInts](#)
 - [ibis::bord](#), [54](#)
 - [ibis::mensa](#), [139](#)
 - [ibis::tabela](#), [222](#)
 - [ibis::table](#), [230](#)
 - [ibis::tabula](#), [244](#)
- [getColumnAsULongs](#)
 - [ibis::bord](#), [55](#)
 - [ibis::tabela](#), [222](#)
 - [ibis::table](#), [230](#)
 - [ibis::tabula](#), [244](#)
- [getColumnAsUShorts](#)
 - [ibis::bord](#), [55](#)
 - [ibis::mensa](#), [139](#)
 - [ibis::tabela](#), [223](#)
 - [ibis::table](#), [230](#)
 - [ibis::tabula](#), [244](#)
- [getCumulativeDistribution](#)
 - [ibis::column](#), [86](#)
 - [ibis::part](#), [167](#)
- [getCurrentRowNumber](#)
 - [ibis::table::cursor](#), [234](#)
- [getDistribution](#)
 - [ibis::column](#), [87](#)
 - [ibis::part](#), [168](#)
- [getFile](#)
 - [ibis::fileManager](#), [112](#)
- [getFileSegment](#)
 - [ibis::fileManager](#), [112](#)
- [getGroup](#)
 - [ibis::resource](#), [206](#)
- [getHistogram](#)
 - [ibis::bord](#), [55](#)
 - [ibis::mensa](#), [139](#)
 - [ibis::tabela](#), [223](#)
 - [ibis::table](#), [230](#)
 - [ibis::tabula](#), [244](#)
- [getHistogram2D](#)
 - [ibis::bord](#), [55](#)
 - [ibis::mensa](#), [140](#)
 - [ibis::tabela](#), [223](#)
 - [ibis::table](#), [231](#)
 - [ibis::tabula](#), [245](#)
- [getHistogram3D](#)
 - [ibis::bord](#), [55](#)
 - [ibis::mensa](#), [140](#)
 - [ibis::tabela](#), [223](#)
 - [ibis::table](#), [231](#)
 - [ibis::tabula](#), [245](#)
- [getHitsAsRanges](#)
 - [ibis::meshQuery](#), [145](#)
- [getHitVector](#)
 - [ibis::query](#), [190](#)

- getInt
 - ibis::bundle, 60
 - ibis::query::result, 194
- getIntArray
 - ibis::column, 87
- getJointDistribution
 - ibis::part, 168
- getMeshShape
 - ibis::part, 169
- getName
 - ibis::selected, 215
- getNumber
 - ibis::resource, 206
- getPointsOnBoundary
 - ibis::meshQuery, 146
- getQualifiedInts
 - ibis::query, 190
- getRIDs
 - ibis::query, 191
- getSerialSize
 - ibis::bitvector, 40
- getString
 - ibis::bundle, 60
 - ibis::column, 87
 - ibis::query::result, 194
- getSum
 - ibis::ambit, 21
 - ibis::bin, 33
 - ibis::direkte, 99
 - ibis::egale, 104
 - ibis::entre, 107
 - ibis::fade, 109
 - ibis::index, 127
 - ibis::keywords, 133
 - ibis::mesa, 143
 - ibis::moins, 148
 - ibis::pack, 151
 - ibis::range, 197
 - ibis::relic, 203
 - ibis::slice, 217
- getUndecidable
 - ibis::column, 87
 - ibis::part, 169
- getValue
 - ibis::resource, 206
- groupby
 - ibis::bord, 55
 - ibis::tabele, 223
 - ibis::table, 231
 - ibis::tabula, 245
- gVerbose
 - ibis, 10
- horometer.h, 270
- ibin.h, 271
- ibis, 3
 - BYTE, 9
 - CATEGORY, 9
 - DOUBLE, 9
 - FLOAT, 9
 - gVerbose, 10
 - init, 9
 - INT, 9
 - LONG, 9
 - OID, 9
 - outerProduct, 9
 - parseQuery, 10
 - SHORT, 9
 - TEXT, 9
 - TYPE_T, 9
 - UBYTE, 9
 - UINT, 9
 - ULONG, 9
 - UNKNOWN_TYPE, 9
 - USHORT, 9
- ibis.h, 272
- ibis::ambit, 19
 - estimate, 21
 - evaluate, 21
 - getSum, 21
 - print, 21
 - read, 21
 - undecidable, 21
 - write, 22
- ibis::bad_alloc, 22
- ibis::bak, 23
 - expandRange, 24
 - print, 24
 - read, 24
 - write, 24
- ibis::bak2, 25
 - expandRange, 26
 - print, 26
 - read, 26
 - write, 26
- ibis::bak2::grain, 27
- ibis::bin, 27
 - binning, 32
 - checkBin, 32
 - estimate, 32, 33
 - evaluate, 33
 - expandRange, 33
 - getSum, 33
 - print, 33
 - read, 34
 - setBoundaries, 34
 - undecidable, 34
 - write, 35
- ibis::bin::granule, 35
- ibis::bitvector, 36
 - adjustSize, 40
 - bitvector, 40
 - clusteringFactor, 40

- flip, 40
- getSerialSize, 40
- markovSize, 40
- operator=, 40
- randomSize, 40
- read, 40
- set, 41
- setBit, 41
- setSize, 41
- write, 41
- ibis::bitvector64, 41
 - adjustSize, 45
 - clusteringFactor, 45
 - flip, 45
 - markovSize, 45
 - randomSize, 45
 - read, 45
 - set, 46
 - setBit, 46
- ibis::bitvector64::const_iterator, 46
- ibis::bitvector64::indexSet, 47
- ibis::bitvector64::iterator, 47
- ibis::bitvector::const_iterator, 48
- ibis::bitvector::indexSet, 48
- ibis::bitvector::iterator, 49
- ibis::bord, 50
 - buildIndex, 53
 - buildIndexes, 53
 - dump, 53
 - estimate, 53
 - getColumnAsBytes, 53
 - getColumnAsDoubles, 53
 - getColumnAsFloats, 53
 - getColumnAsInts, 54
 - getColumnAsLongs, 54
 - getColumnAsShorts, 54
 - getColumnAsStrings, 54
 - getColumnAsUBytes, 54
 - getColumnAsUInts, 54
 - getColumnAsULongs, 55
 - getColumnAsUSHORTS, 55
 - getHistogram, 55
 - getHistogram2D, 55
 - getHistogram3D, 55
 - groupby, 55
 - indexSpec, 56
 - orderby, 56
- ibis::bord::column, 56
 - selectDoubles, 57
 - selectLongs, 57
- ibis::bundle, 58
 - getInt, 60
 - getString, 60
 - rowCounts, 60
- ibis::bundle0, 61
- ibis::bundle1, 61
- ibis::bundles, 63
 - reorder, 64
 - truncate, 64
- ibis::bylt, 65
 - bylt, 66
 - evaluate, 66
 - print, 66
 - read, 66, 67
 - write, 67
- ibis::category, 67
- ibis::colDoubles, 68
- ibis::colDoubles
 - reorder, 70
 - sort, 70
- ibis::colFloats, 70
- ibis::colFloats
 - reorder, 72
 - sort, 72
- ibis::colInts, 72
- ibis::colInts
 - reorder, 74
 - sort, 74
- ibis::colLongs, 75
- ibis::colLongs
 - reorder, 76
 - sort, 76
- ibis::colUInts, 77
- ibis::colUInts
 - reorder, 78
 - sort, 78
- ibis::colULongs, 79
- ibis::colULongs
 - reorder, 80
 - sort, 80
- ibis::column, 81
 - actualMinMax, 85
 - append, 85
 - column, 85
 - computeMinMax, 85, 86
 - dataFileName, 86
 - estimateRange, 86
 - evaluateRange, 86
 - findString, 86
 - getActualMax, 86
 - getActualMin, 86
 - getCumulativeDistribution, 86
 - getDistribution, 87
 - getIntArray, 87
 - getString, 87
 - getUndecidable, 87
 - selectBytes, 87
 - selectDoubles, 87
 - selectLongs, 87
 - selectShorts, 87
 - selectValues, 88
 - truncateData, 88
- ibis::column::indexLock, 88
- ibis::column::info, 88

- ibis::column::mutexLock, 89
- ibis::column::writeLock, 89
- ibis::colValues, 90
- ibis::colValues
 - reorder, 92
 - sort, 92
- ibis::compRange, 92
- ibis::compRange::barrel, 94
- ibis::compRange::barrel
 - recordVariable, 95
- ibis::dictionary, 95
 - find, 96
 - insertRaw, 96
- ibis::direkte, 96
 - estimate, 98
 - evaluate, 99
 - getSum, 99
 - print, 99
 - read, 99
 - undecidable, 99
- ibis::discretePoisson, 100
- ibis::discretePoisson1, 100
- ibis::discreteZipf, 101
- ibis::discreteZipf1, 101
- ibis::discreteZipf2, 101
- ibis::egale, 102
 - estimate, 104
 - evaluate, 104
 - getSum, 104
 - print, 104
 - read, 104
 - undecidable, 105
 - write, 105
- ibis::entre, 105
 - estimate, 107
 - evaluate, 107
 - getSum, 107
 - print, 107
 - write, 107
- ibis::fade, 107
 - evaluate, 109
 - getSum, 109
 - print, 109
 - write, 109
- ibis::fileManager, 110
- ibis::fileManager
 - ACCESS_PREFERENCE, 112
 - getFile, 112
 - getFileSegment, 112
 - recordPages, 112
 - tryGetFile, 113
- ibis::fileManager::cleaner, 113
- ibis::fileManager::readLock, 113
- ibis::fileManager::roFile, 114
- ibis::fileManager::roFile
 - score, 114
- ibis::fileManager::storage, 115
- ibis::fileManager::storage
 - inUse, 116
 - pastUse, 116
 - unnamed, 116
- ibis::fuzz, 116
 - evaluate, 118
 - fuzz, 118
 - print, 118
 - read, 118
 - write, 118
- ibis::horometer, 119
 - resume, 119
 - start, 119
 - stop, 119
- ibis::index, 120
 - addBins, 125
 - addBits, 125
 - create, 125
 - divideCounts, 126
 - estimate, 126, 127
 - evaluate, 127
 - expandRange, 127
 - getSum, 127
 - index, 125
 - indexFileName, 127
 - isIndex, 127
 - mapValues, 128
 - print, 128
 - read, 129
 - setBases, 129
 - str, 130
 - sumBins, 129
 - sumBits, 129, 130
 - undecidable, 130
 - write, 130
- ibis::index::barrel, 130
- ibis::keywords, 131
 - estimate, 133
 - evaluate, 133
 - getSum, 133
 - keywords, 133
 - print, 133
 - read, 134
 - readKeyword, 134
 - readTermDocFile, 134
- ibis::lessi, 134
- ibis::mensa, 135
 - buildIndex, 138
 - buildIndexes, 138
 - dump, 138
 - estimate, 138
 - getColumnAsBytes, 138
 - getColumnAsDoubles, 138
 - getColumnAsInts, 138
 - getColumnAsLongs, 139
 - getColumnAsShorts, 139
 - getColumnAsUBytes, 139

- getColumnAsUInts, 139
- getColumnAsUShorts, 139
- getHistogram, 139
- getHistogram2D, 140
- getHistogram3D, 140
- indexSpec, 140
- orderby, 140
- ibis::MersenneTwister, 141
- ibis::mesa, 141
 - estimate, 143
 - evaluate, 143
 - getSum, 143
 - print, 143
 - undecidable, 143
 - write, 144
- ibis::meshQuery, 144
- ibis::meshQuery
 - bitvectorToCoordinates, 145
 - getHitsAsRanges, 145
 - getPointsOnBoundary, 146
- ibis::moins, 146
 - estimate, 148
 - evaluate, 148
 - getSum, 148
 - print, 148
 - write, 148
- ibis::nameList, 149
- ibis::nameList
 - find, 149
- ibis::pack, 150
 - estimate, 151
 - evaluate, 151
 - getSum, 151
 - print, 151
 - read, 152
 - undecidable, 152
 - write, 152
- ibis::pale, 152
 - estimate, 154
 - evaluate, 154
 - print, 154
 - read, 154
 - undecidable, 154
 - write, 155
- ibis::part, 155
 - append, 165
 - appendToBackup, 165
 - commit, 165
 - doCompare, 165, 166
 - doScan, 166
 - estimateMatchAny, 166
 - evaluateJoin, 166
 - get1DDistribution, 167
 - getColumn, 167
 - getCumulativeDistribution, 167
 - getDistribution, 168
 - getJointDistribution, 168
 - getMeshShape, 169
 - getUndecidable, 169
 - loadIndex, 169
 - lookforString, 169
 - matchAny, 169
 - metaTags, 169
 - part, 165
 - purgeIndexFiles, 169
 - readTDC, 169
 - reorder, 170
 - rollback, 170
 - setMeshShape, 170
 - setMetaTags, 170
- ibis::part::advisoryLock, 170
- ibis::part::barrel, 171
 - read, 172
 - seek, 172
- ibis::part::cleaner, 172
- ibis::part::info, 172
- ibis::part::mutexLock, 173
- ibis::part::readLock, 173
- ibis::part::vault, 174
 - read, 175
 - tellReal, 175
- ibis::part::writeLock, 175
- ibis::qAnyAny, 175
- ibis::qAnyAny
 - printRange, 176
- ibis::qContinuousRange, 176
- ibis::qContinuousRange
 - inRange, 178
 - printRange, 178
- ibis::qExpr, 178
- ibis::qExpr
 - printRange, 180
 - separateSimple, 180
 - simplify, 180
 - TYPE, 180
- ibis::qExpr::weight, 181
- ibis::qRange, 181
- ibis::qRange
 - inRange, 182
- ibis::qString, 182
- ibis::query, 183
 - contractQuery, 189
 - countDeltaPairs, 189
 - countEqualPairs, 189
 - countHits, 190
 - estimate, 190
 - evaluate, 190
 - expandQuery, 190
 - getHitVector, 190
 - getQualifiedInts, 190
 - getRIDs, 191
 - isValidToken, 191
 - limit, 191
 - logMessage, 191

- orderby, 191
- orderPairs, 191
- printSelected, 192
- printSelectedWithRID, 192
- processJoin, 192
- query, 189
- removeComplexConditions, 192
- sequentialScan, 192
- setSelectClause, 192
- setTable, 192
- setWhereClause, 193
- sortEquiJoin, 193
- sortRangeJoin, 193
- ibis::query::result, 193
 - getInt, 194
 - getString, 194
 - reset, 195
- ibis::range, 195
 - estimate, 197
 - evaluate, 197
 - expandRange, 197
 - getSum, 197
 - print, 197
 - read, 198
 - undecidable, 198
 - write, 198
- ibis::rangeJoin, 198
- ibis::relic, 199
 - append, 202
 - construct, 202
 - estimate, 202, 203
 - evaluate, 203
 - getSum, 203
 - print, 203
 - read, 203
 - relic, 202
 - write, 204
- ibis::resource, 204
 - add, 206
 - getGroup, 206
 - getNumber, 206
 - getValue, 206
 - operator[], 206
 - parseNameValuePairs, 206
 - read, 206
 - write, 207
- ibis::rid_t, 207
- ibis::rid_t::name, 208
- ibis::ridHandler, 208
- ibis::ridHandler
 - append, 209
 - read, 209
 - write, 209
- ibis::roster, 209
 - locate, 211
 - mergeBlock2, 211
 - roster, 210
 - writeSorted, 211
- ibis::sapid, 211
 - evaluate, 212
 - print, 212
 - write, 212
- ibis::sbiad, 212
 - evaluate, 213
 - print, 214
 - write, 214
- ibis::selected, 214
 - find, 215
 - getName, 215
- ibis::slice, 215
 - estimate, 217
 - evaluate, 217
 - getSum, 217
 - print, 217
 - read, 217, 218
 - write, 218
- ibis::tabele, 218
 - buildIndex, 220
 - buildIndexes, 221
 - dump, 221
 - estimate, 221
 - getColumnAsBytes, 221
 - getColumnAsDoubles, 221
 - getColumnAsFloats, 221
 - getColumnAsInts, 221
 - getColumnAsLongs, 222
 - getColumnAsShorts, 222
 - getColumnAsStrings, 222
 - getColumnAsUBytes, 222
 - getColumnAsUInts, 222
 - getColumnAsULongs, 222
 - getColumnAsUShorts, 223
 - getHistogram, 223
 - getHistogram2D, 223
 - getHistogram3D, 223
 - groupby, 223
 - indexSpec, 224
 - orderby, 224
- ibis::table, 224
 - buildIndex, 228
 - buildIndexes, 228
 - create, 228
 - dump, 228
 - estimate, 228
 - getColumnAsBytes, 228
 - getColumnAsDoubles, 229
 - getColumnAsFloats, 229
 - getColumnAsInts, 229
 - getColumnAsLongs, 229
 - getColumnAsShorts, 229
 - getColumnAsStrings, 230
 - getColumnAsUBytes, 230
 - getColumnAsUInts, 230
 - getColumnAsULongs, 230

- getColumnAsUShorts, 230
- getHistogram, 230
- getHistogram2D, 231
- getHistogram3D, 231
- groupby, 231
- indexSpec, 231
- orderby, 232
- parseNames, 232
- ibis::table::cursor, 232
 - fetch, 233
 - getColumnAsByte, 234
 - getCurrentRowNumber, 234
- ibis::table::row, 234
- ibis::tableList, 235
- ibis::tableList
 - add, 236
 - end, 236
 - operator[], 237
 - remove, 237
- ibis::tablex, 237
 - append, 238
 - appendRow, 238
 - appendRows, 239
 - readCSV, 239
 - write, 239
- ibis::tabula, 240
 - buildIndex, 242
 - buildIndexes, 242
 - dump, 242
 - estimate, 242
 - getColumnAsBytes, 242
 - getColumnAsDoubles, 243
 - getColumnAsFloats, 243
 - getColumnAsInts, 243
 - getColumnAsLongs, 243
 - getColumnAsShorts, 243
 - getColumnAsStrings, 243
 - getColumnAsUBytes, 244
 - getColumnAsUInts, 244
 - getColumnAsULongs, 244
 - getColumnAsUShorts, 244
 - getHistogram, 244
 - getHistogram2D, 245
 - getHistogram3D, 245
 - groupby, 245
 - indexSpec, 245
 - orderby, 245
- ibis::tafel, 246
 - append, 248
 - appendRow, 248
 - appendRows, 248
 - readCSV, 248
 - write, 249
- ibis::text, 249
 - append, 250
 - findString, 250
 - readString, 251
 - selectUIInts, 251
 - startPositions, 251
- ibis::uniformRandomNumber, 251
- ibis::util::counter, 252
- ibis::util::ioLock, 252
- ibis::util::mutexLock, 253
- ibis::util::quietLock, 253
- ibis::util::readLock, 253
- ibis::util::writeLock, 254
- ibis::zona, 254
 - evaluate, 255
 - print, 255
 - read, 256
 - write, 256
 - zona, 255
- ibis::zone, 256
 - estimate, 257
 - evaluate, 258
 - print, 258
 - read, 258
 - undecidable, 258
 - write, 258
- idirekte.h, 272
- ikywords.h, 273
- index
 - ibis::index, 125
- index.h, 273
- indexFileName
 - ibis::index, 127
- indexSpec
 - ibis::bord, 56
 - ibis::mensa, 140
 - ibis::tabelle, 224
 - ibis::table, 231
 - ibis::tabula, 245
- init
 - ibis, 9
- inRange
 - ibis::qContinuousRange, 178
 - ibis::qRange, 182
- insert
 - array_t, 18
- insertRaw
 - ibis::dictionary, 96
- INT
 - ibis, 9
- inUse
 - ibis::fileManager::storage, 116
- irelic.h, 274
- iroster.h, 274
- isIndex
 - ibis::index, 127
- isValidToken
 - ibis::query, 191
- keywords
 - ibis::keywords, 133

- limit
 - ibis::query, 191
- loadIndex
 - ibis::part, 169
- locate
 - ibis::roster, 211
- logMessage
 - ibis::query, 191
- LONG
 - ibis, 9
- lookforString
 - ibis::part, 169
- mapValues
 - ibis::index, 128
- markovSize
 - ibis::bitvector, 40
 - ibis::bitvector64, 45
- matchAny
 - ibis::part, 169
- mena.h, 275
- mergeBlock2
 - ibis::roster, 211
- meshQuery.h, 275
- metaTags
 - ibis::part, 169
- nosharing
 - array_t, 18
- OID
 - ibis, 9
- operator=
 - ibis::bitvector, 40
- operator[]
 - array_t, 18
 - ibis::resource, 206
 - ibis::tableList, 237
- orderBy
 - ibis::bord, 56
 - ibis::mena, 140
 - ibis::query, 191
 - ibis::tabele, 224
 - ibis::table, 232
 - ibis::tabula, 245
- orderPairs
 - ibis::query, 191
- outerProduct
 - ibis, 9
- parseNames
 - ibis::table, 232
- parseNameValuePairs
 - ibis::resource, 206
- parseQuery
 - ibis, 10
- part
 - ibis::part, 165
- part.h, 276
- pastUse
 - ibis::fileManager::storage, 116
- predicate.h, 277
- PREFERRED_BLOCK_SIZE
 - const.h, 269
- print
 - ibis::ambit, 21
 - ibis::bak, 24
 - ibis::bak2, 26
 - ibis::bin, 33
 - ibis::bylt, 66
 - ibis::direkte, 99
 - ibis::egale, 104
 - ibis::entre, 107
 - ibis::fade, 109
 - ibis::fuzz, 118
 - ibis::index, 128
 - ibis::keywords, 133
 - ibis::mesa, 143
 - ibis::moins, 148
 - ibis::pack, 151
 - ibis::pale, 154
 - ibis::range, 197
 - ibis::relic, 203
 - ibis::sapid, 212
 - ibis::sbiad, 214
 - ibis::slice, 217
 - ibis::zona, 255
 - ibis::zone, 258
- printRange
 - ibis::qAnyAny, 176
 - ibis::qContinuousRange, 178
 - ibis::qExpr, 180
- printSelected
 - ibis::query, 192
- printSelectedWithRID
 - ibis::query, 192
- processJoin
 - ibis::query, 192
- purgeIndexFiles
 - ibis::part, 169
- qExpr.h, 278
- query
 - ibis::query, 189
- query.h, 279
- randomSize
 - ibis::bitvector, 40
 - ibis::bitvector64, 45
- read
 - ibis::ambit, 21
 - ibis::bak, 24
 - ibis::bak2, 26
 - ibis::bin, 34
 - ibis::bitvector, 40

- ibis::bitvector64, 45
- ibis::bylt, 66, 67
- ibis::direkte, 99
- ibis::egale, 104
- ibis::fuzz, 118
- ibis::index, 129
- ibis::keywords, 134
- ibis::pack, 152
- ibis::pale, 154
- ibis::part::barrel, 172
- ibis::part::vault, 175
- ibis::range, 198
- ibis::relic, 203
- ibis::resource, 206
- ibis::ridHandler, 209
- ibis::slice, 217, 218
- ibis::zona, 256
- ibis::zone, 258
- readCSV
 - ibis::tablex, 239
 - ibis::tafel, 248
- readKeyword
 - ibis::keywords, 134
- readString
 - ibis::text, 251
- readTDC
 - ibis::part, 169
- readTermDocFile
 - ibis::keywords, 134
- recordPages
 - ibis::fileManager, 112
- recordVariable
 - ibis::compRange::barrel, 95
- relic
 - ibis::relic, 202
- remove
 - ibis::tableList, 237
- removeComplexConditions
 - ibis::query, 192
- reorder
 - ibis::bundles, 64
 - ibis::colDoubles, 70
 - ibis::colFloats, 72
 - ibis::colInts, 74
 - ibis::colLongs, 76
 - ibis::colUInts, 78
 - ibis::colULongs, 80
 - ibis::colValues, 92
 - ibis::part, 170
- reserve
 - array_t, 18
- reset
 - ibis::query::result, 195
- resource.h, 280
- resume
 - ibis::horometer, 119
- rids.h, 280
- rollback
 - ibis::part, 170
- roster
 - ibis::roster, 210
- rowCounts
 - ibis::bundle, 60
- score
 - ibis::fileManager::roFile, 114
- seek
 - ibis::part::barrel, 172
- selectBytes
 - ibis::column, 87
- selectDoubles
 - ibis::bord::column, 57
 - ibis::column, 87
- selectLongs
 - ibis::bord::column, 57
 - ibis::column, 87
- selectShorts
 - ibis::column, 87
- selectUInts
 - ibis::text, 251
- selectValues
 - ibis::column, 88
- separateSimple
 - ibis::qExpr, 180
- sequentialScan
 - ibis::query, 192
- set
 - ibis::bitvector, 41
 - ibis::bitvector64, 46
- setBases
 - ibis::index, 129
- setBit
 - ibis::bitvector, 41
 - ibis::bitvector64, 46
- setBoundaries
 - ibis::bin, 34
- setMeshShape
 - ibis::part, 170
- setMetaTags
 - ibis::part, 170
- setSelectClause
 - ibis::query, 192
- setSize
 - ibis::bitvector, 41
- setTable
 - ibis::query, 192
- setWhereClause
 - ibis::query, 193
- SHORT
 - ibis, 9
- simplify
 - ibis::qExpr, 180
- size
 - array_t, 18

- sort
 - ibis::colDoubles, 70
 - ibis::colFloats, 72
 - ibis::colInts, 74
 - ibis::colLongs, 76
 - ibis::colUInts, 78
 - ibis::colULongs, 80
 - ibis::colValues, 92
- sortEquiJoin
 - ibis::query, 193
- sortRangeJoin
 - ibis::query, 193
- stableSort
 - array_t, 18, 19
- start
 - ibis::horometer, 119
- startPositions
 - ibis::text, 251
- std, 10
- stop
 - ibis::horometer, 119
- str
 - ibis::index, 130
- sumBins
 - ibis::index, 129
- sumBits
 - ibis::index, 129, 130
- tab.h, 281
- table.h, 281
- tafel.h, 283
- tellReal
 - ibis::part::vault, 175
- TEXT
 - ibis, 9
- THREAD_RWLOCK_INITIALIZER
 - const.h, 269
- topk
 - array_t, 19
- truncate
 - ibis::bundles, 64
- truncateData
 - ibis::column, 88
- tryGetFile
 - ibis::fileManager, 113
- twister.h, 283
- TYPE
 - ibis::qExpr, 180
- TYPE_T
 - ibis, 9
- UBYTE
 - ibis, 9
- UINT
 - ibis, 9
- ULONG
 - ibis, 9
- undecidable
 - ibis::ambit, 21
 - ibis::bin, 34
 - ibis::direkte, 99
 - ibis::egale, 105
 - ibis::index, 130
 - ibis::mesa, 143
 - ibis::pack, 152
 - ibis::pale, 154
 - ibis::range, 198
 - ibis::zone, 258
- UNKNOWN_TYPE
 - ibis, 9
- unnamed
 - ibis::fileManager::storage, 116
- USHORT
 - ibis, 9
- util.h, 284
- write
 - ibis::ambit, 22
 - ibis::bak, 24
 - ibis::bak2, 26
 - ibis::bin, 35
 - ibis::bitvector, 41
 - ibis::bylt, 67
 - ibis::egale, 105
 - ibis::entre, 107
 - ibis::fade, 109
 - ibis::fuzz, 118
 - ibis::index, 130
 - ibis::mesa, 144
 - ibis::moins, 148
 - ibis::pack, 152
 - ibis::pale, 155
 - ibis::range, 198
 - ibis::relic, 204
 - ibis::resource, 207
 - ibis::ridHandler, 209
 - ibis::sapid, 212
 - ibis::sbiad, 214
 - ibis::slice, 218
 - ibis::tablex, 239
 - ibis::tafel, 249
 - ibis::zona, 256
 - ibis::zone, 258
- writeSorted
 - ibis::roster, 211
- zona
 - ibis::zona, 255