This project requires the implementation of a **database catalog** or **metadata**. The catalog contains data about the objects in the database, e.g., tables, attributes, views, indexes, triggers, etc. The catalog is used by many components of a database system. The query parser checks that all the tables/attributes in a query exist in the database and that the attributes are used correctly in expressions, e.g., you do not divide a string by an integer. This is called semantics analysis. The query optimizer extracts statistics on tables and attributes from the catalog in order to compute the optimal query execution plan. The statistics are created/updated during a statistics maintenance operation executed at certain time intervals.

To keep matters simple, the catalog in this project contains only data on **tables** and their corresponding **attributes**. For each table, the **name**, the **number of tuples**, and the **location of the file containing the data** are stored. For each attribute, the **name**, **type**, and **number of distinct values** are stored.

Catalog Class

The project package includes class `Catalog`. The interface is given in `Catalog.h`. Catalog reads all the data from a SQLite database file at startup. All the subsequent operations are executed on the memory-resident data structures. When the system is stopped, the catalog content is materialized to disk in the same SQLite database, so that, when the database is started again, the new content is preserved. Saving to the database can be also triggered at other time instants.

Class `Attribute` is a container for the elements of a table attribute. These are read from the SQLite database. When a table is created, only the name and type of attributes have to be specified. The number of tuples and of distinct elements in each attribute is updated directly in the SQLite database. We will provide these values later in the project.

Class `Schema` is a container for all the `Attributes` in a table. There is a `Schema` object for every table in the database. The order of the attributes in the schema requires care. It has to be identical to the order in which attributes are stored in the physical record representation. It is recommended to store the position of an attribute in the database and, when the `Catalog` is read, create the schema following the order of the attributes in the database.

`Config.h` includes global constants and definitions used across the entire project.

`Swap.h` includes two macros for swapping classes that implement the assignment operator (=) and `STL` objects, respectively. `Swap` moves the content from one object to another, in each direction.

Also included in the project package, a series of generic data structures might be helpful for implementing the functionality of the catalog. Their use is entirely optional. `TwoWayList` is a doubly-linked list. `InefficientMap` and `EfficientMap` are two alternative implementations for a map (hash table) data structure. `InefficientMap` is optimal for relatively small-size maps up to 1000 keys, while `EfficientMap` becomes optimal above that size. `Keyify` and `Swapify` classes are templates to create objects that can be used by `TwoWayList`, `InefficientMap`, and `EfficientMap` from any primitive data type. A few examples are provided. `TwoWayList` requires `Swapify` (it actually requires a `Swap` method), while `InefficientMap` and `EfficientMap` require `Keyify`. To be precise, `InefficientMap` requires an `IsEqual` method, while `EfficientMap` requires both `IsEqual` and `LessThan`.

`main.cc` contains sample code that uses `Catalog` class.

`makefile` contains the compilation definitions. To compile the code, type `make` at the terminal in the project package directory. The executable `main.out` is generated.

Requirements

- Create a SQLite database `catalog.sqlite`. Inside the database, create the necessary tables to store the catalog data on tables and attributes.

- Implement the interface of class `Catalog` as defined in `Catalog.h`. Write your code in `Catalog.cc`. You are responsible for declaring all the data structures and interfacing with the SQLite database. Additionally, you can create new methods in class `Catalog`. However, the interface of the class cannot be modified.

- Write a menu-based console application (in `main.cc`) that allows the user to at least create a table, drop a table, display the content of the catalog, and save the content of the catalog to the database. It can also include any other functionality you may found useful.

- Ideally, all your code has to be written in two files: `Catalog.cc` and `main.cc`. There is no need to do it differently, but if you plan so, discuss with the instructor/TA.

- Required packages: `libsqlite3-0`, `libsqlite3-dev`, `sqlite3`, `sqlite3-doc`.

Resources

- `http://www.sqlite.org/cintro.html`

- `http://www.tutorialspoint.com/sqlite/sqlite_c_cpp.htm`

- `http://www.tpc.org/tpch/default.asp`