NITCbase

A relational database management system

Prepared by Cliford Joshy

Some DBMS theory

- Relational Database – collection of persistent data, organized into tables.
- Relational Database Management System (DBMS) – software system that supports creation, population, and querying on a relational database
- Relational Database Management System (RDBMS)
  - Data is organized into tables, also known as relations.
  - A table is a collection of rows (called records or tuples). Each row represents a data entity.
  - Each tuple can have a number of subfields, called columns (or fields or attributes).
  - Columns define the names and types of the data in each of the subfields of a row.
  - The number of columns per tuple is fixed for a given table.

The relation **Students(id, name, gpa)** is shown here.

- id, name and gpa are the attributes of the relation.
- The relation has 4 tuples corresponding to 4 students.
- The value corresponding to a particular attribute is of the same type in every record. That is, every id is an integer, every name is a string and so on.

id	name	gpa
5001	Dave	9.4
5002	Tina	8.6
5003	Alan	7
5004	Anne	9.1

What is NITCbase?

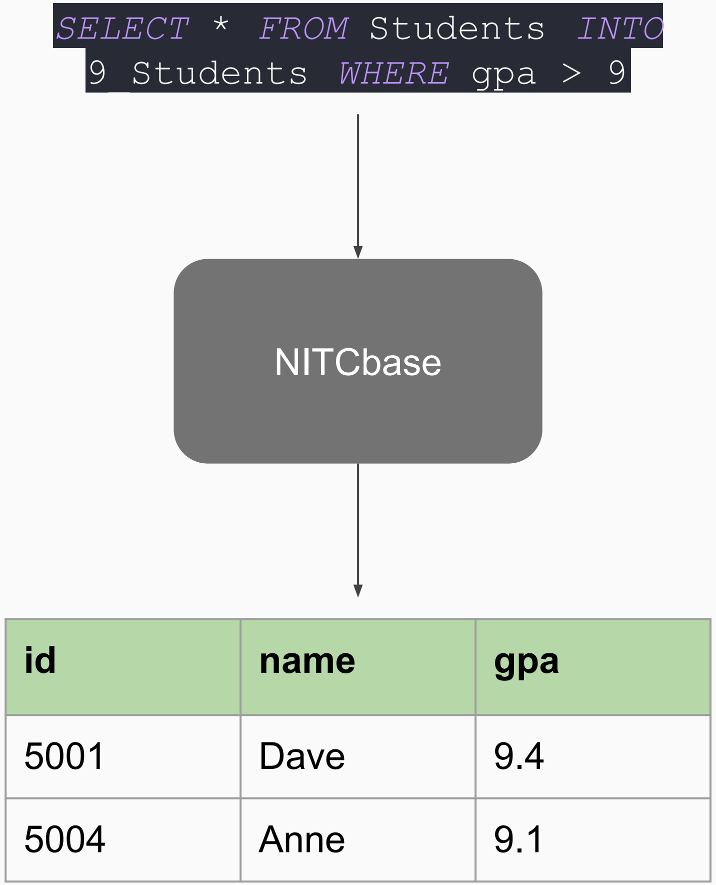
NITCbase is an elementary RDBMS with minimal features. It is designed to help you understand the working of a DBMS by implementing one yourself.

The design of the system simplifies tedious implementation work by proposing some core abstractions. The design and the core algorithms are given to you.

The NITCbase design is object-oriented.

Abstractions

- Only two types of data are allowed; NUMBER and STRING.
  - Both types are fixed to have a size of 16 bytes
- All the data is stored in a simulated disk environment
  - The “disk” contains 8192 blocks with each block having a size of 2048 bytes.
  - The entire space occupied by data in NITCbase cannot exceed 16 MB
- All DB queries happen exclusively through command line interfaces made available to the user.



```
SELECT * FROM Students INTO
9_Students WHERE gpa > 9
```

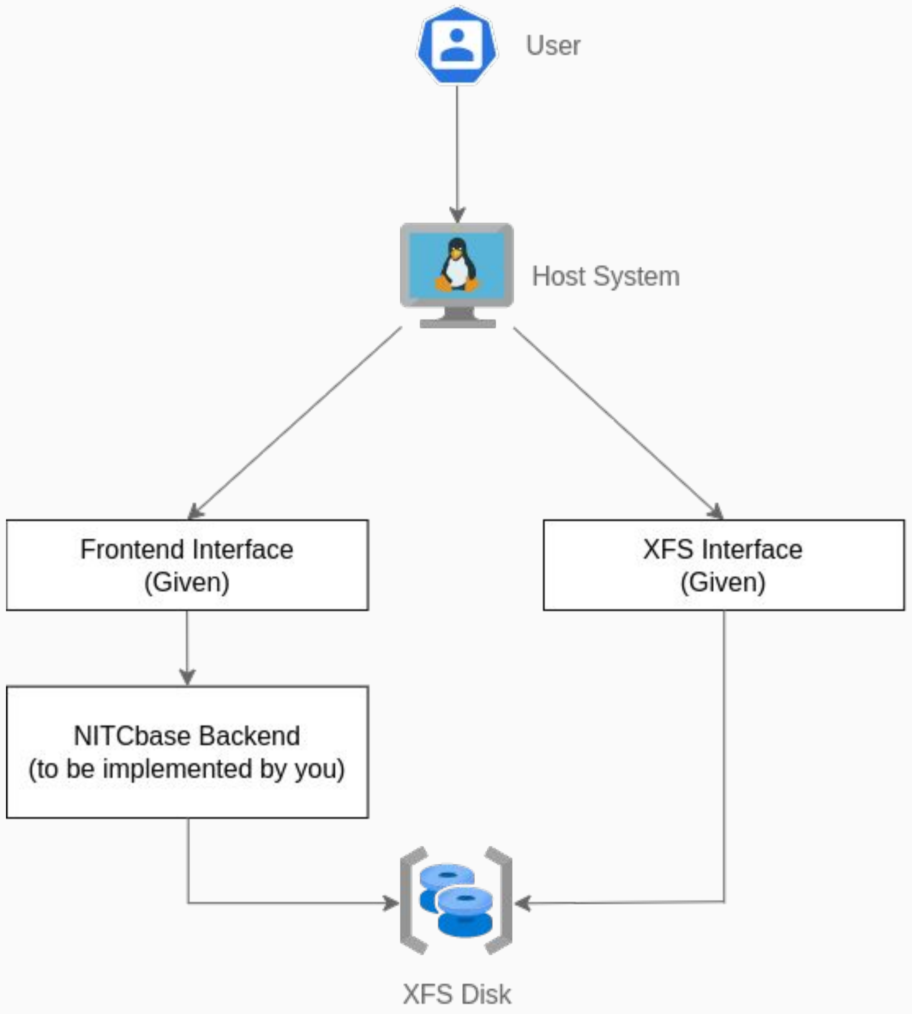
id	name	gpa
5001	Dave	9.4
5004	Anne	9.1

9_students

A high level look at what your database will do.

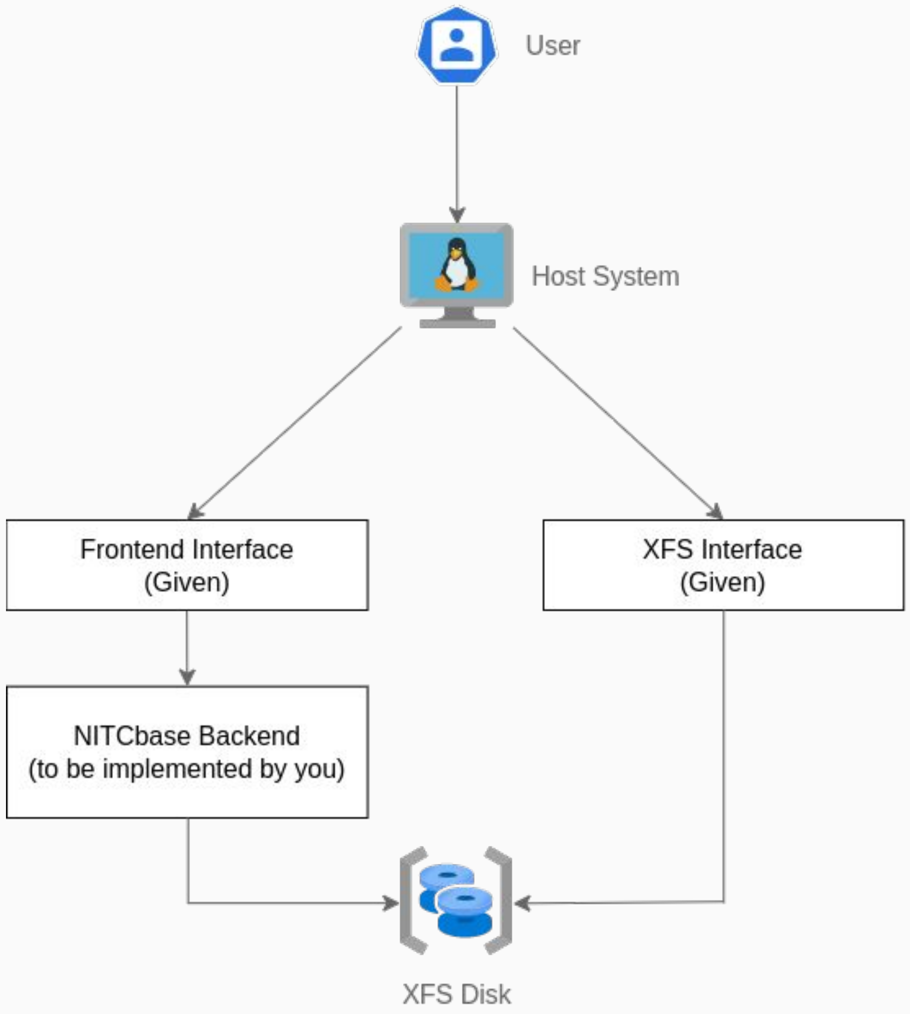
Components of the Project

1. NITCbase
2. XFS Interface


User
Host System
Frontend Interface (Given)
XFS Interface (Given)
NITCbase Backend (to be implemented by you)
XFS Disk

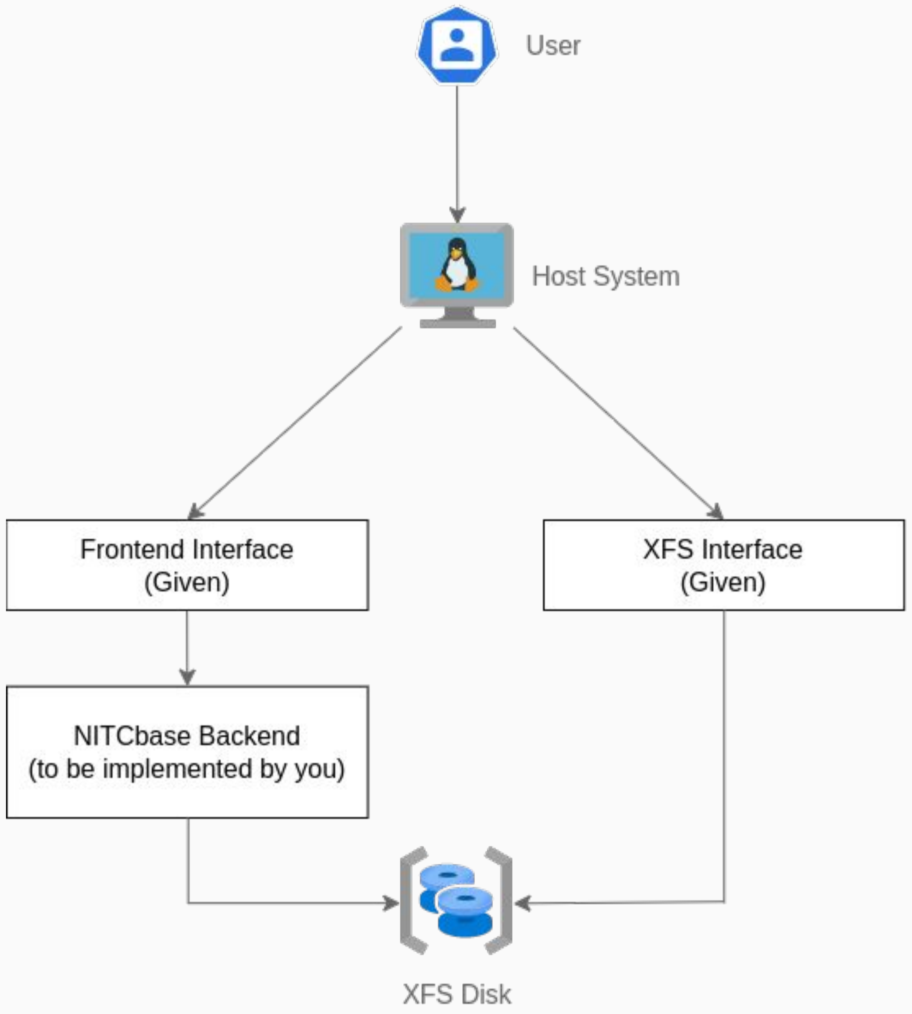
NITCbase

- Frontend Interface: parses high level commands from the user. Provided to you.
- Disk Interface : Methods to read/write data blocks to the disk. Provided to you.
- Database Backend: Implements the functionality. For you to do


User
Host System
Frontend Interface (Given)
XFS Interface (Given)
NITCbase Backend (to be implemented by you)
XFS Disk

XFS Interface

- An external interface to the disk.
- A full implementation of NITCbase functionality - Fully implemented and given to you.
- All database operations can be performed from your host machine.
- Provided to aid you in your own implementation.


User
Host System
Frontend Interface (Given)
XFS Interface (Given)
NITCbase Backend (to be implemented by you)
XFS Disk

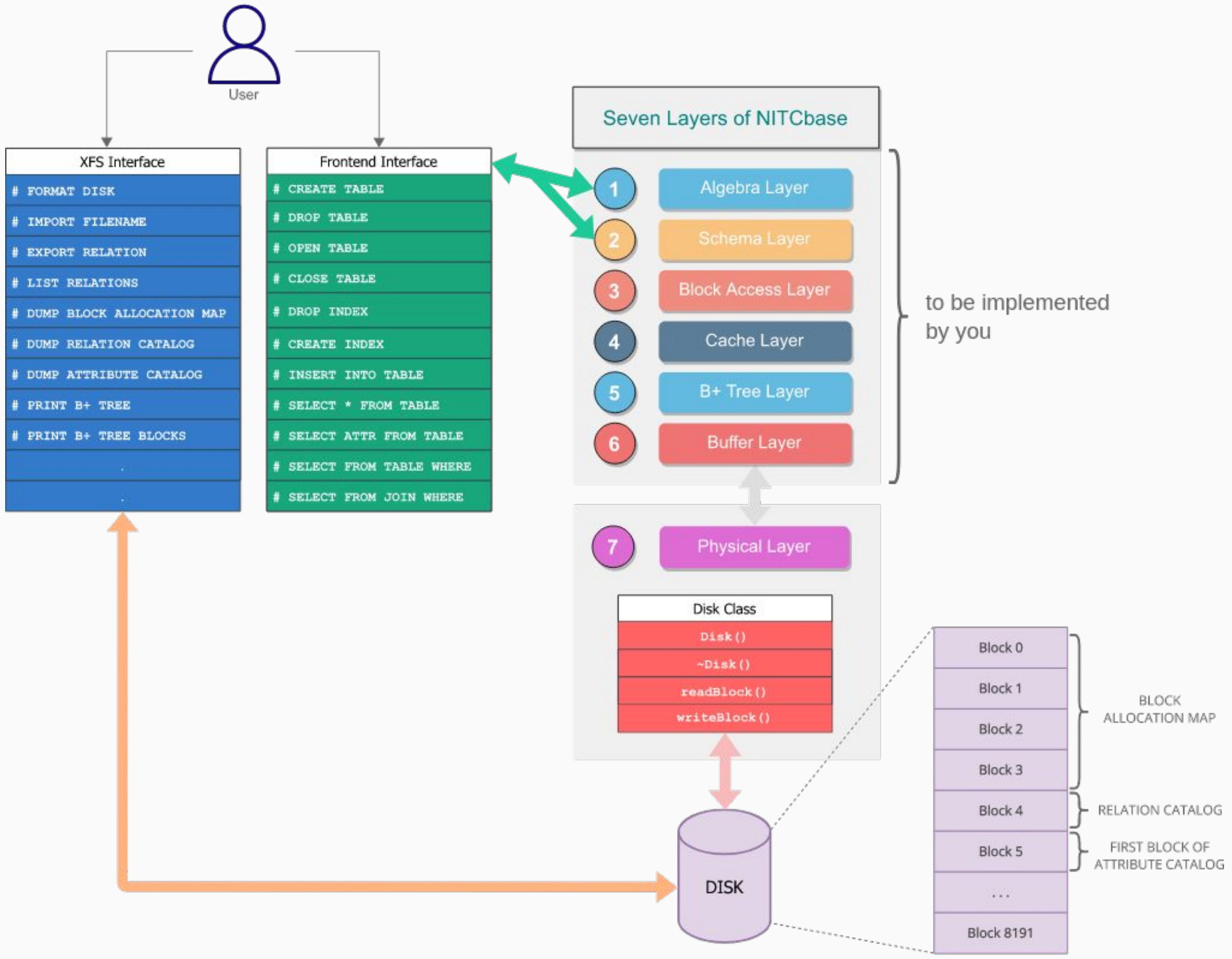
What is on the disk?

- The NITCbase design treats a block on the XFS disk in one of two ways; either a record block or an index block.
- A record block stores the data (raw tuples corresponding of each relation).
- An index block stores information that facilitates the efficient access of data blocks during search queries on the database.
- There are also reserved blocks (storing relation catalog /attribute catalog / disk free list) used to store metadata relating to relations and the disk itself.

NITCbase design

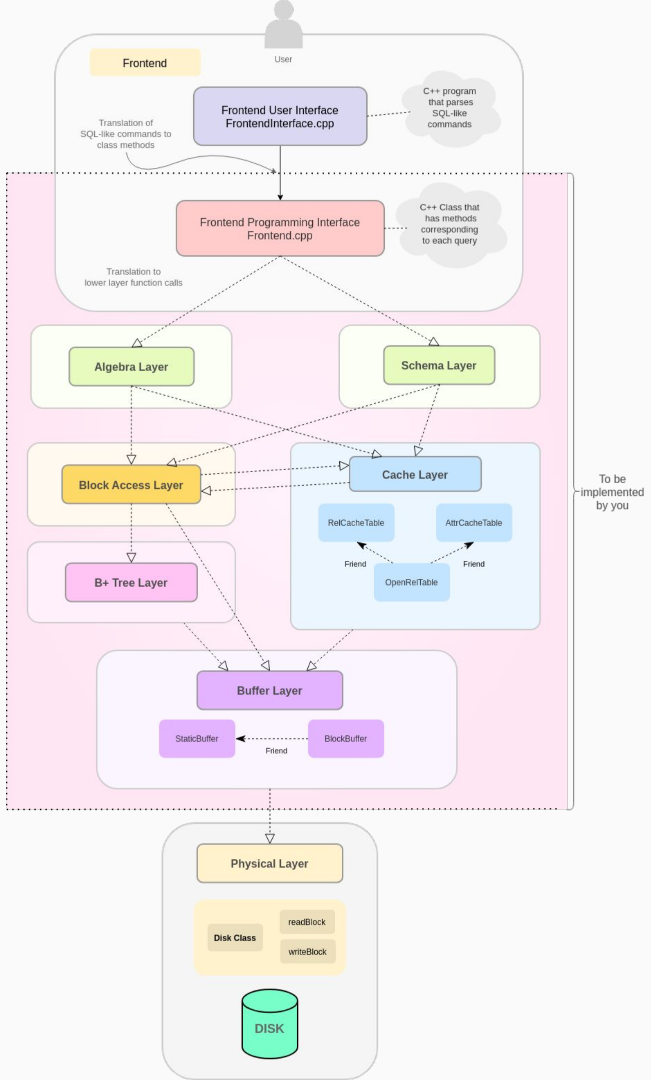
NITCbase has an eight layer design where each layer implements some subset of the requirements using lower layer functionality. The layers are as follows:

1. Frontend Interface: parses high level commands and invokes Algebra/Scheme layer functions.
2. Algebra Layer: handles algebraic operations (select, project and join queries).
3. Schema Layer: handles schema operations (create/delete/rename relations)
4. Block Access Layer: handles common disk block access operations (tuple insertion, search)
5. B+ Tree Layer: handles all indexing operations for efficient data access and retrieval.
6. Cache Layer: handles relation metadata caching (cache of open relations, catalogs etc.)
7. Buffer Layer: handles low level buffering of disk blocks for efficient disk data access
8. Physical Layer: handles abstractions for disk access.


User
XFS Interface
# FORMAT DISK
# IMPORT FILENAME
# EXPORT RELATION
# LIST RELATIONS
# DUMP BLOCK ALLOCATION MAP
# DUMP RELATION CATALOG
# DUMP ATTRIBUTE CATALOG
# PRINT B+ TREE
# PRINT B+ TREE BLOCKS
.
.
Frontend Interface
# CREATE TABLE
# DROP TABLE
# OPEN TABLE
# CLOSE TABLE
# DROP INDEX
# CREATE INDEX
# INSERT INTO TABLE
# SELECT * FROM TABLE
# SELECT ATTR FROM TABLE
# SELECT FROM TABLE WHERE
# SELECT FROM JOIN WHERE
Seven Layers of NITCbase
1 Algebra Layer
2 Schema Layer
3 Block Access Layer
4 Cache Layer
5 B+ Tree Layer
6 Buffer Layer
to be implemented by you
7 Physical Layer
Disk Class
Disk()
~Disk()
readBlock()
writeBlock()
DISK
Block 0
Block 1
Block 2
Block 3
Block 4
Block 5
...
Block 8191
BLOCK ALLOCATION MAP
RELATION CATALOG
FIRST BLOCK OF ATTRIBUTE CATALOG

High Level Interactions

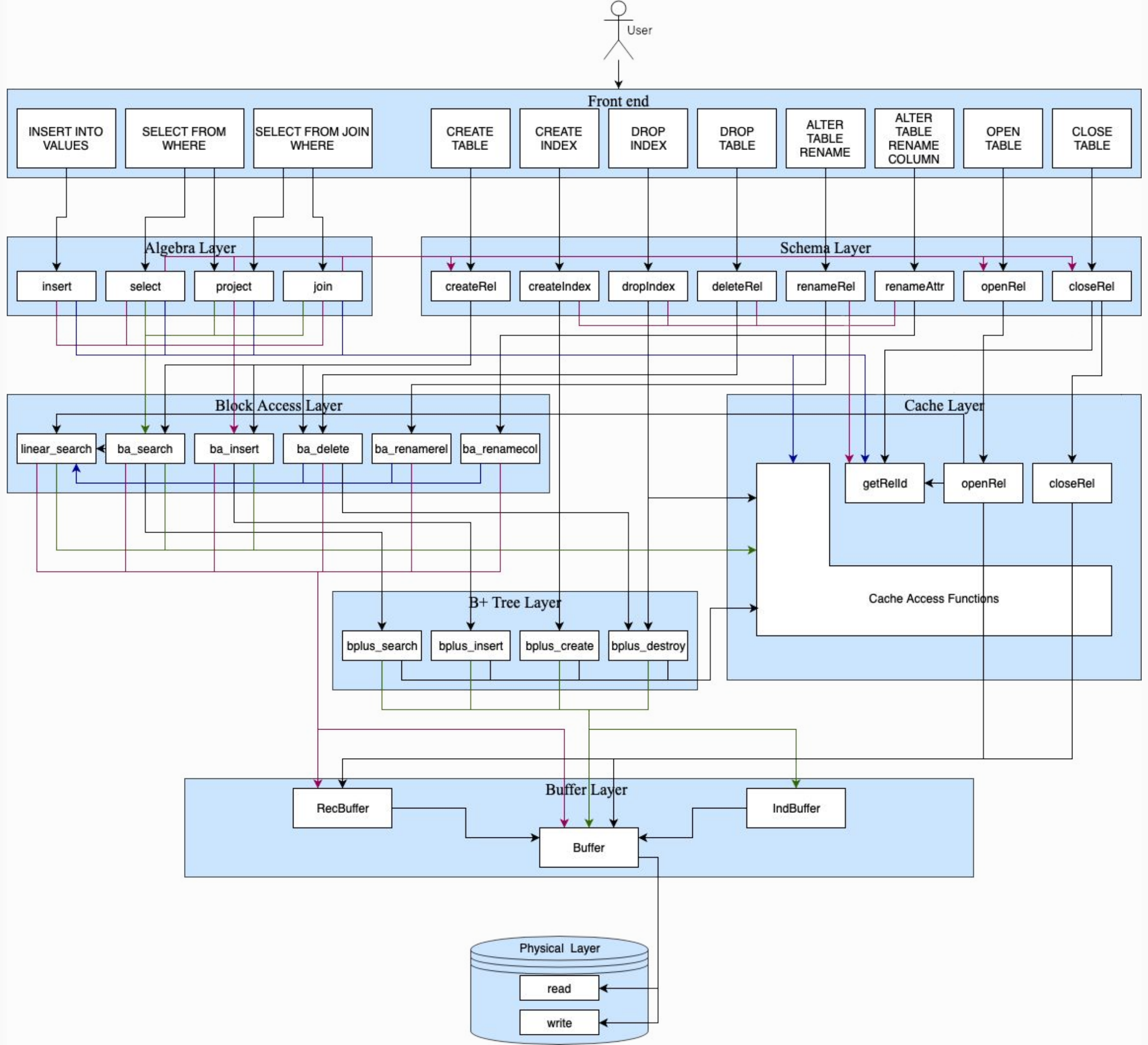
- Each layer uses the methods made available by the lower layers to implement it's functionality.
- Implementing the 6 layers between the Frontend Interface and the Physical Layer is your task.
- These layers comprise the core functionality of a relational DBMS



User
Frontend
Frontend User Interface FrontendInterface.cpp
C++ program that parses SQL-like commands
Translation of SQL-like commands to class methods
Frontend Programming Interface Frontend.cpp
C++ Class that has methods corresponding to each query
Translation to lower layer function calls
Algebra Layer
Schema Layer
Block Access Layer
Cache Layer
RelCacheTable
AttrCacheTable
Friend
Friend
OpenRelTable
B+ Tree Layer
Buffer Layer
StaticBuffer
Friend
BlockBuffer
To be implemented by you
Physical Layer
Disk Class
readBlock
writeBlock
DISK

Good Luck!

Pictured on the right is every major function and the calls between them that you will have implemented by the end of this project



Proceed to:

https://nitcbase.github.io