Project 2: Database System Develop

> Designed by Teaching group
>
> Demo is provided by ZHANG Ziyang
>
> Presentation time in the 16th week

Project Introduction:

In response to the Ministry of Education's "101 Plan" which emphasizes **core competency development in database systems**, and to promote a shift in teaching from **“using databases” to “building databases”**, this project focuses on database kernel implementation. Through **modular experimental tasks**, students will gain a deep understanding of the core principles and engineering practices of Database Management Systems (DBMS), cultivating system-level development capabilities.

This project is based on a provided **Java framework** (with some core modules already implemented), and involves developing a simple database system, including modules such as: SQL parsing, logical operators, physical operators, storage layer design, and data transformation. Students are required to comprehend the framework and complete or extend the modules based on provided content and code samples. The framework currently includes the following functionalities:

- Disk I/O read/write (storage layer design)
- Support for `CREATE TABLE`, `INSERT`, `UPDATE` statements
- Equality filtering (`WHERE`) and logical operations (`AND`)
- Full table scans (`SELECT *` and `SeqScan` operations)

Students may use the provided project structure or build their own entirely new structure, as long as the requirements of the tasks are met.

Data Preparation

For the project presentation, at least one test table must be created with more than 30 rows of data, for example:

Create table

```
create table t( id int, name char, age int, gpa float);
```

Insert

```sql
insert into t (id, name, age, gpa) values (1, 'a', 18, 3.6);
insert into t (id, name, age, gpa) values (2, 'b', 19, 3.65);
insert into t (id, name, age, gpa) values (3, 'abb', 18, 3.86);
insert into t (id, name, age, gpa) values (4, 'abc', 19, 2.34);
insert into t (id, name, age, gpa) values (5, 'ef', 20, 3.25);
insert into t (id, name, age, gpa) values (6, 'bbc', 21, 3.20);
```

Select

```sql
select * from t;
select * from t where t.age = 19;
```

Task 1: Storage Management (20 points)

1. Page replacement policy (Least Recently Used - LRU) algorithm (10 points)
   - Complete the `Victim()`, `Pin(int frameId)`, and `Unpin(int frameId)` methods in `LRUReplacer`, and pass the `LRUReplacerTest` JUnit tests.
2. Implement actual disk I/O and binary file storage (10 points)

Task 2: Query Processing (50 points)

1. Basic: SQL Statement Implementation (40 points)

The current teaching framework supports the following features, which students may extend:

1. **Table Management**
   - Support for `CREATE TABLE` statements with at least the following data types:
     - `INT` (integer)
     - `VARCHAR` (variable-length string)
     - `DOUBLE` (double precision float)
   - Tables must be persistently stored on disk to ensure data is retained after system restarts.
2. **Data Operations**
   - Support for `INSERT` and `UPDATE` operations.
   - At least **30 records** should be inserted and maintained to test system stability and storage management.
   - All data modification operations (insert, delete, update) must be written to disk in real-time to ensure durability.

1.1 Basic DML Operations (20 points)

For all operations below, whether successful or failed, program execution must not be affected and appropriate logs should be generated.

- Support `show tables` command. Expected output:

  ```
  show tables;
  ```

  ```
  21:29:38.281 INFO: |-----------|
  21:29:38.281 INFO: |  Tables   |
  21:29:38.281 INFO: |-----------|
  21:29:38.300 INFO: |     t     |
  21:29:38.301 INFO: |     a     |
  21:29:38.301 INFO: |-----------|
  ```

- Support `describe table` command. Returns table name, column names, and data types. If the table is not found, an exception should be thrown or an error logged.

  ```
  describe t;
  ```

  ```
  21:35:18.211 INFO: |-------------------------|
  21:35:18.212 INFO: |    Field    |   Type    |
  21:35:18.212 INFO: |    id       |   int     |
  21:35:18.212 INFO: |    name     |   char    |
  21:35:18.212 INFO: |    age      |   int     |
  21:35:18.212 INFO: |    gpa      |   float   |
  21:35:18.212 INFO: |-------------------------|
  ```

- Support `DROP TABLE` operation with proper handling and logging whether the table exists or not.
- Support `EXPLAIN` for query plan display. Example:

  ```
  explain select t.id, t.name from t where t.age>18;
  ```

  Output:

  ```
  21:38:18.602 INFO: ProjectOperator(selectItems=[t.id, t.name])
  └── LogicalFilterOperator(condition=t.age > 18)
          └── TableScanOperator(table=t)
  ```

1.2 Logical and Physical Operators (20 points)

- Support projection operations (column selection with `SELECT`, i.e., `ProjectOperator`), allowing arbitrary column selection instead of just `SELECT *`.

```
select t.id, t.name from t where t.age > 18;
```

```
21:39:20.082 INFO: ┌────────────────┬────────────────┐
21:39:20.099 INFO: │     t.id       │     t.name     │
21:39:20.099 INFO: ├────────────────┼────────────────┤
21:39:20.143 INFO: │       4        │      abc       │
21:39:20.143 INFO: ├────────────────┼────────────────┤
21:39:20.143 INFO: │       6        │      bbc       │
21:39:20.144 INFO: └────────────────┴────────────────┘
```

- Implement full table scan and conditional filtering (WHERE clause parsing), supporting:
  - Any column
  - `AND`, `OR` logic
  - Equality and range queries: `=`, `>`, `>=`, `<`, `<=`

  Example:

```
select * from t1 where col1 = xxx or col2 = yyy;
select * from t1 where col1 > xxx;
select * from t1 where col1 > xxx and col2 = yyy;
```

- Support `DELETE` operations, with full support for conditions (AND, OR, equality, and range filters). (10 points)

---

2. Advanced: Join Operators and Advanced SeqScan Calculations (10 points)

- Support aggregation functions: `SUM()`, `MAX()`, `MIN()`, `COUNT()`
- Support `GROUP BY`, `ORDER BY`
- Implement `Nested Loop Join` for equi-joins
- Support `IN`, `NOT IN`, `EXISTS`
- Implement query optimizer to generate different query plans for different SQL statements
- Support partial `ALTER TABLE` operations

Task 3: Optimizer (20 points)

1. Basic: Sequential Scan Implementation (SeqScan) (10 points)

- Understand the provided `SeqScan` module
- Be able to explain its implementation and execution logic in detail
- Implementation details will be evaluated during the presentation

2. Advanced: Index Optimization Implementation (10 points)

1. Index Support:
    - Implement in-memory B+ Tree or Hash Index
    - Optimizer must choose appropriate index and generate corresponding query plans
2. Validation:
    - Use large enough datasets for testing
    - Ensure correctness and performance of index functionality

Note: This task focuses on students' understanding and implementation of core components of the query optimizer, requiring both theoretical knowledge and practical engineering skills.

Task 4: Presentation (10 points)

1. A complete interface for command input and result display
2. Clear and effective classroom presentation and expression skills
3. Completion within the allotted time
4. No team member is late or causes delay

Other Advanced Implementations

1. Dynamic Hash Expansion:
    - Implement dynamic bucket splitting
    - Automatically adjust hash table capacity
2. Conflict Handling:
    - Design efficient disk page conflict resolution for hashing
    - Ensure data access efficiency
3. Persistent Storage:
    - Store index structures on disk
    - Ensure index persistence after system restart
    - Optimize disk I/O performance

Note: These advanced features are highly challenging and will be important for evaluation, reflecting the technical depth and innovation of the project.