Report

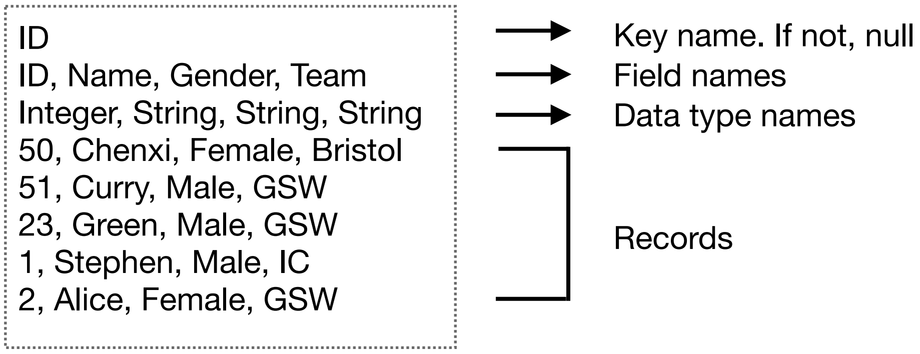
Before Start

What I did

1. All the basic requirement.
2. Types
3. Interface
4. Catalogs
5. Transactions
6. User interface, which can give error messages for invalid command.
7. Complex query language. (support using SELECT specific column /JOIN/WHERE at the same time.)

Important tips:

1. When you use my query class, it needs some time (10 seconds or more), because in DBLAB/TEST there is a very big database (200000 records) to test my database.
2. The following picture explains the structure of the txt file of the table.



```
ID
ID, Name, Gender, Team
Integer, String, String, String
50, Chenxi, Female, Bristol
51, Curry, Male, GSW
23, Green, Male, GSW
1, Stephen, Male, IC
2, Alice, Female, GSW
```

It is strongly recommended that to start this program from the Query part. Although this is for the extension, it is easy to test my code on a user side.

Query

The query supports 11 different commands

(ALL queries should be in the capital. You can copy my example command):

1. SHOW DATABASES;

2. USE "database name" (You can change the database anytime you want!);

```
USE NBA;
USE PET;
```

3. SHOW TABLES;

4. SELECT fieldName1, fieldName2.. FROM tableName WHERE compareFiledName "logic Symbol" compareValue;
   Tip: ”logic symbol" could be ">" "=" "<" “><"

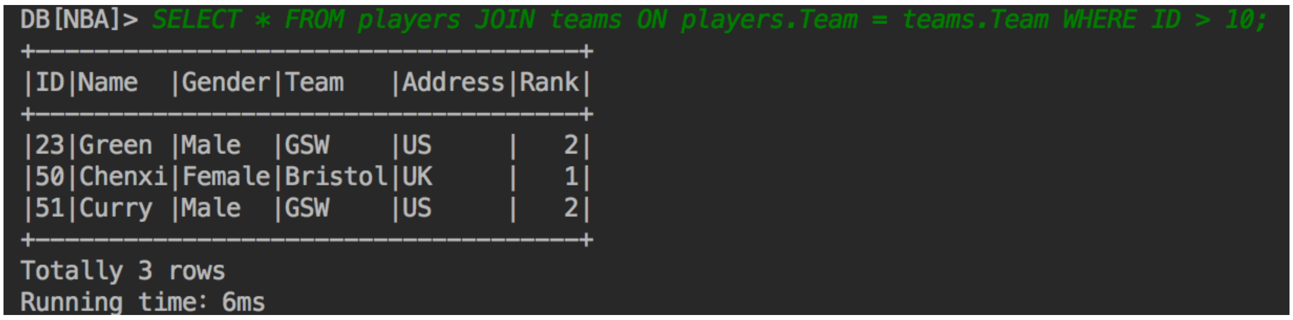
```
SELECT * FROM players;
```

SELECT ID, Team FROM players WHERE ID > 30;

5. JOIN

Before using this command, please use SHOW DATABASES; -> USE NBA;

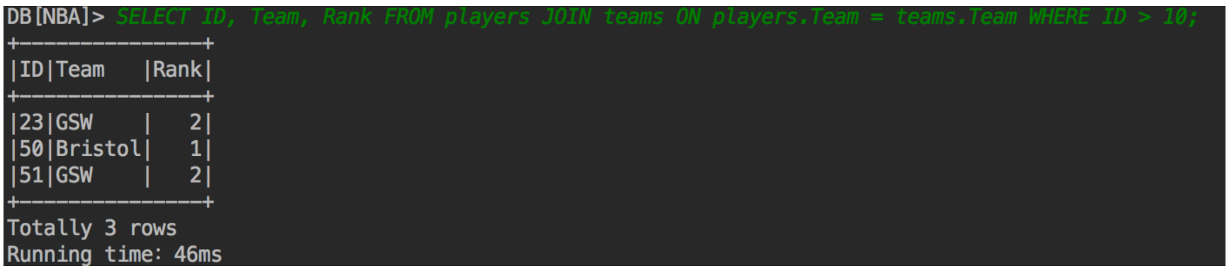
SELECT * FROM players JOIN teams ON players.Team = teams.Team WHERE ID > 10;



```
DB[NBA]> SELECT * FROM players JOIN teams ON players.Team = teams.Team WHERE ID > 10;
+-------------------------------------+
|ID|Name  |Gender|Team   |Address|Rank|
+-------------------------------------+
|23|Green |Male  |GSW    |US     |   2|
|50|Chenxi|Female|Bristol|UK     |   1|
|51|Curry |Male  |GSW    |US     |   2|
+-------------------------------------+
Totally 3 rows
Running time: 6ms
```

SELECT ID, Team, Rank FROM players JOIN teams ON players.Team = teams.Team WHERE ID > 10;

(This is the most complicated command I support.)



```
DB[NBA]> SELECT ID, Team, Rank FROM players JOIN teams ON players.Team = teams.Team WHERE ID > 10;
+---------------+
|ID|Team   |Rank|
+---------------+
|23|GSW    |   2|
|50|Bristol|   1|
|51|GSW    |   2|
+---------------+
Totally 3 rows
Running time: 46ms
```

6. UPDATE tableName SET fieldName = newValue WHERE compareFiledName "logic Symbol" compareValue;

UPDATE players SET Team = Bristol WHERE Name = Alice;

7. DELETE FROM tableName WHERE compareFiledName "logic Symbol" compareValue;

DELETE FROM players WHERE ID = 2;

8. SAVE tableName; (This is not a standard MySQL query)

SAVE players;

9. CREATE TABLE tableName (FieldName1 Type NOT/KEY, FieldName2 Type NOT/KEY...);

CREATE TABLE pets (Name String NOT, ID Integer KEY, Owner String NOT);

10. INSERT INTO tableName (FieldName1, FieldNam2..) VALUES (value1, value2....)

INSERT INTO pets (Name, ID, Owner) VALUES (Dog, 1, Harry);

11. QUIT

QUIT;

Explanation of the data structure

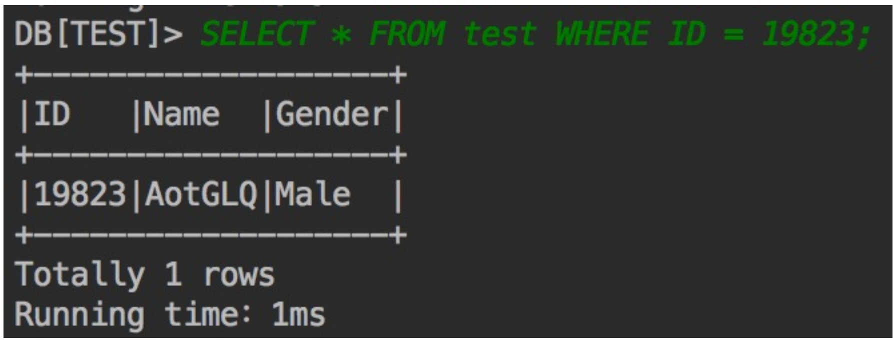
My database is based on treeMap.

If a user chooses not to add a key, then row numbers will be a key. If a user chooses to add a key then the key's value will be a key.

	KEY (in treeMap)	Value
Table with key	key (like id, name)	Record
Table without key	row number	Record

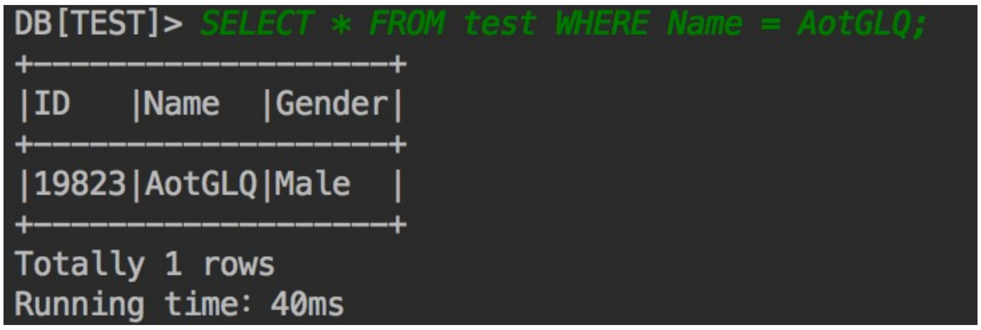
So if a user sets a key and searches by a key, it will be considerably fast. You can try the following commands. I will show the difference between search by a key and not by a key.
Please use the following commands.

SHOW DATABASES;
USE TEST;
SELECT * FROM test;   (This need some time. Totally 200000 rows.)
SELECT * FROM test WHERE ID = 19823;   (ID is key. In my laptop, it needs 1ms)



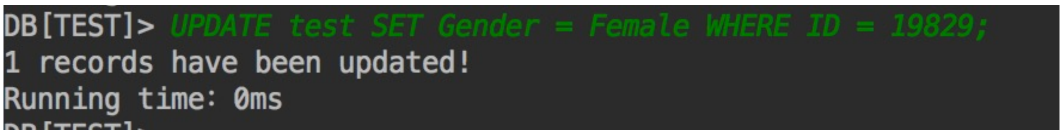
```
DB[TEST]> SELECT * FROM test WHERE ID = 19823;
+-------------------+
|ID   |Name  |Gender|
+-------------------+
|19823|AotGLQ|Male  |
+-------------------+
Totally 1 rows
Running time: 1ms
```

SELECT * FROM test WHERE Name = AotGLQ;   (Name is not key. It needs 40ms)



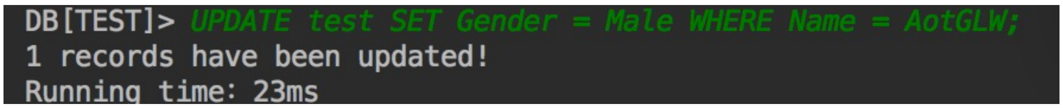
```
DB[TEST]> SELECT * FROM test WHERE Name = AotGLQ;
+-------------------+
|ID   |Name  |Gender|
+-------------------+
|19823|AotGLQ|Male  |
+-------------------+
Totally 1 rows
Running time: 40ms
```

UPDATE test SET Gender = Female WHERE ID = 19829;   (ID is key. It needs 1ms)



```
DB[TEST]> UPDATE test SET Gender = Female WHERE ID = 19829;
1 records have been updated!
Running time: 0ms
```

UPDATE test SET Gender = Male WHERE Name = AotGLW;   (Name is not key. It needs 23ms)



```
DB[TEST]> UPDATE test SET Gender = Male WHERE Name = AotGLW;
1 records have been updated!
Running time: 23ms
```

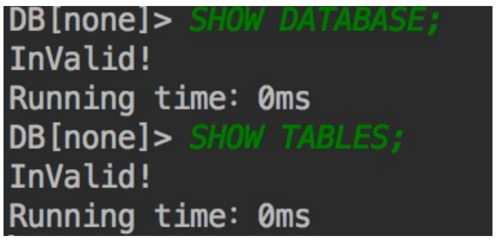
Conclusion

If you search by a key, then the time complexity is O(logN). It is relatively quick. If you don't search by a key, then the time complexity is O(N). It is much slower.
The following table shows that why I choose a tree map. I didn't consider hash map because of memory.

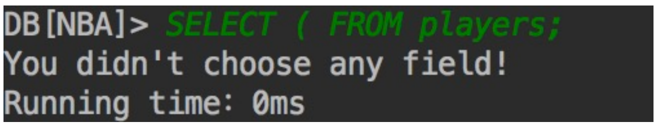
	Time complexity(TreeMap)	**Time complexity(Linked list)**	**Time complexity(Array List)**
select by key	O(logN)	O(N)	O(N)
select not by key	O(N)	O(N)	O(N)
insert	O(logN)	O(1)	O(1)
delete by key	O(logN)	O(N) Need O(N) to search, O(1) to delete	O(N) Need O(N) to search, O(N) to delete
update by key	O(logN)	O(N)	O(N)
update not by key	O(N)	O(N)	O(N)
join	O(N*logN)	O(N*N)	O(N*N)

INVALID COMMAND TEST

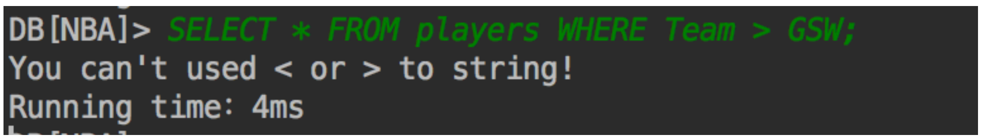
My parser and code have abilities to detect both logic and syntax errors.
1.Syntax error test: You could copy these code and also could try your own illegal inputs.
a. SHOW DATABASE; (typo, it should be DATABASES)
b. SHOW TABLES; (Without using a database, it is illegal.)


```
DB[none]> SHOW DATABASE;
InValid!
Running time: 0ms
DB[none]> SHOW TABLES;
InValid!
Running time: 0ms
```

c. SHOW DATABASES; -> USE NBA; ->SELECT (FROM players; (You didn't choose any field!)


```
DB[NBA]> SELECT ( FROM players;
You didn't choose any field!
Running time: 0ms
```

d. SELECT * FROM players WHERE Team > GSW; (My logic operation doesn't support > and <)

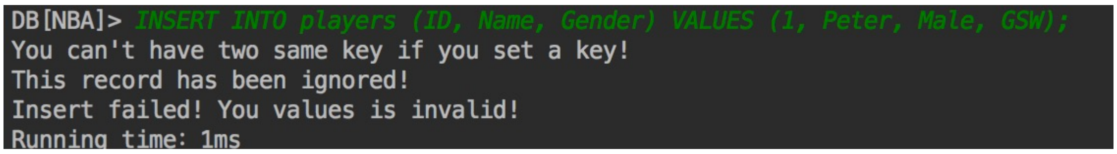

```
DB[NBA]> SELECT * FROM players WHERE Team > GSW;
You can't used < or > to string!
Running time: 4ms
```

2. Logic error test:

Before running. Please enter the following code:

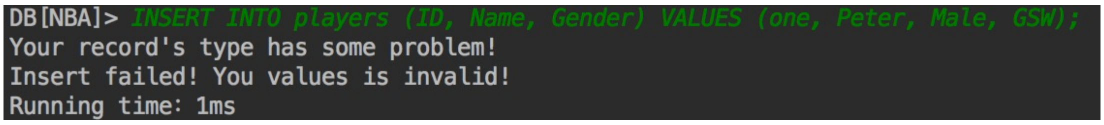
SHOW DATABASES; -> USE NBA; -> SELECT * FROM players;

a. You can't insert two records has the same key if your table has a key.
INSERT INTO players (ID, Name, Gender) VALUES (1, Peter, Male, GSW);
(ERROR: You can't have two same key if you set a key!)



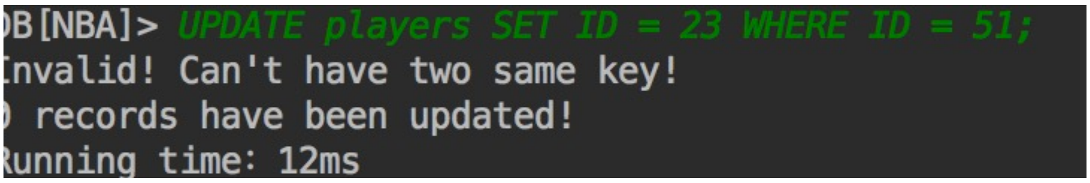
```
DB[NBA]> INSERT INTO players (ID, Name, Gender) VALUES (1, Peter, Male, GSW);
You can't have two same key if you set a key!
This record has been ignored!
Insert failed! You values is invalid!
Running time: 1ms
```

b. You can't insert a record with wrong types
INSERT INTO players (ID, Name, Gender) VALUES (one, Peter, Male, GSW);
(ERROR: Your record's type has some problem!)



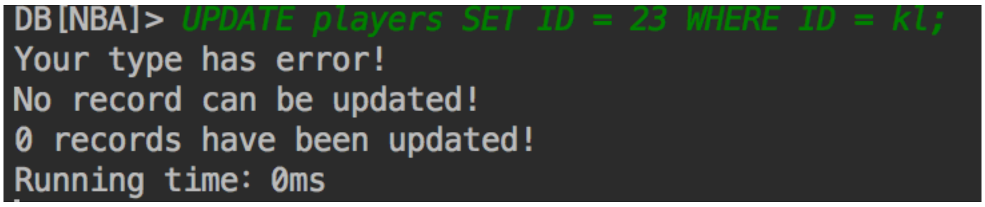
```
DB[NBA]> INSERT INTO players (ID, Name, Gender) VALUES (one, Peter, Male, GSW);
Your record's type has some problem!
Insert failed! You values is invalid!
Running time: 1ms
```

c. You can't update a record if you change the key and the key already exist.
UPDATE players SET ID = 23 WHERE ID = 51;
(ERROR: Invalid! Can't have two same key!)



```
B[NBA]> UPDATE players SET ID = 23 WHERE ID = 51;
nvalid! Can't have two same key!
 records have been updated!
unning time: 12ms
```

d. You can't update a record's field with wrong types.
UPDATE players SET ID = 23 WHERE ID = kl;
(ERROR: Your type has an error!)



```
DB[NBA]> UPDATE players SET ID = 23 WHERE ID = kl;
Your type has error!
No record can be updated!
0 records have been updated!
Running time: 0ms
```

Generally speaking, my parser can deal with most illegal inputs.

Go Back To Very First

Considering the time complexity and the memory, finally I choose tree map as the basic data structure. In my design, I have 7 different classes, which are Record, Table, Databases, Query and PrintTable, Type, Logic.

Record

This is the basic data cell. It is an ArrayList.

Table
A table has many Record objects. These records are put into a Tree Map. A table is the most important part. It has methods to select, delete. update, insert and join.

Database
It is a set of table. It has many tables. Also, it is used to deal with the relationship between tables.

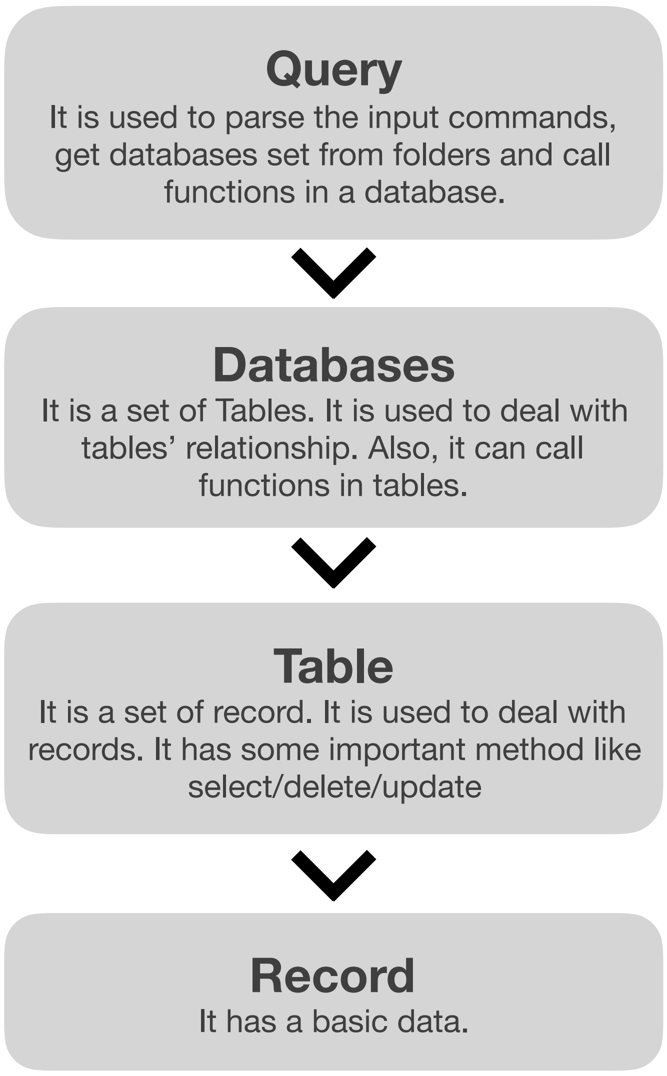
Query
It is a set of databases. Also, it is a parser which could parse the command inputs by users.

PrintTable
Since printing table is always used in Database and Query, so I have this class to make code DRY.

Type
This is used to check types like Integer, String, Long. It can also check whether an argument is empty or not. Generally speaking, it is like a police.

Logic
In my design, I support 4 different kinds of logical operation(>, <, = , ><) for Integer and Long. The function supports 2 different kinds of logical operation(=, ><). So I have this class to check and pass the logic arguments.

Class Structure


Query
It is used to parse the input commands, get databases set from folders and call functions in a database.
Databases
It is a set of Tables. It is used to deal with tables' relationship. Also, it can call functions in tables.
Table
It is a set of record. It is used to deal with records. It has some important method like select/delete/update
Record
It has a basic data.