Course Project Description

In this project, you are going to design and implement a small-scale relational database management system (DBMS).

•***Data****:* any information worth preserving, most likely in electronic form.
•***Database****:* a collection of data, organized for access and modification, preserved over a long period.
•***Query****:* an operation that extracts specified data from the database.
•***Relation****:* an organization of data into a two-dimensional table, where rows (tuples) represent basic entities or facts of some sort, and columns (attributes) represent properties of those entities.
•***Schema****:* a description of the structure of the data in a database, often called "metadata."

Specifically you are asked to:

1. Allow the creation and the deletion of schemas for the database tables. The schema will contain the name of the table, the columns of the table , and the allowable range of domains for columns (constraints) of the table, etc.

2. Allow the insertion and the deletion of records.

3. Allow the update of values of the attributes of records.

4. Allow the retrieval of data that may involve one or more tables.

5. Deny or abort any operations that would cause the schema integrity constraints to be violated.

6. Report syntactic, or semantic errors involved in any (SQL) commands.

For simplicity, the DBMS needs to handle only one user at a time and you can assume that you will never have more data than you can store in main memory. Also, this project does not require the DBMS to maintain a log

Some commercial DBMSs store the whole database as a single huge file, which involves complex physical data management. In this project, you can store each table in a file for easy maintenance.

Schema

A schema (table definition) will have a fixed structure with the following information: table name, column name, column type, and domain constraints. An example of the schema could be:

Relation	Attribute Name	Attribute Type	Domain Constraints
test	student_name	char(15)	student_name != “”
test	student_id	int	student_id !=0
test	student_age	int	(student_age > 10) AND (student_age < 70)

Possible types for an attribute

int	Integer
char(strlen)	Character string of length str_len
decimal	Real Number

An attribute name must be alphanumeric (starting with an alphabetic character), have a length of at most 256 characters, and must accept the underscore character (_).

Use a special file named db/schema to store the database schema. For each relation, the file schema has a line beginning with that relation name, in which attribute names alternate with types. The character # separates elements of these lines. A schema can contain more dynamic information such as number of tuples in the relation
For example, the schema file might contain lines such as

```
Students#id#INT#name#STR#dept#STR
Depts#name#STR#office#STR
```

Use the file system to store the relations.

For example, the table Students (id, name, dept) would be stored in the text file students. The file Students has one line for each tuple. Values of components of a tuple are stored as character strings, separated by the special marker character #. For instance, the file Students might look like:

```
123#POOJA#CS
345#TINA#ME
111#RAJ#CH
110#RAJLAXMI#CH
108#RAHUL#CS
```

Data Definition and Data Manipulation Commands
You are required to process the following data definition and data manipulation commands as well as the help commands listed at the end of the table below:

Command	**Command Structure**
Create Table	CREATE TABLE table_name (attr_name1 attr_type1, attr_name2 attr_type2, …)
Drop Table	DROP TABLE table_name
DESCRIBE T_NAME	Describes the schema of table T_NAME.
Insert	INSERT INTO table_name VALUES(attr_value1, attr_value2, …) Note that NULL is not permitted for any attribute.

Delete	DELETE FROM table_name WHERE condition_list
Update	UPDATE table_name SET attr1_name = attr1_value, attr2_name = attr2_value… WHERE condition_list
Select	SELECT attr_list FROM table_list WHERE condition_list
HELP TABLES	Prints out the list of tables defined.
HELP CMD	Describes the built-in command CMD, where CMD is one of the data definition and manipulation commands listed above.
QUIT	Quits the program.

Development Guideline

Language

You can use Java, C or C++ to implement the DBMS. Your command parser can use the native features of your language of implementation. Note that you are not allowed to use any database related libraries (e.g. JDBC for Java) of the language of your choice or an existing DBMS for data manipulation. You should store all persistent data in files.

Command parsing

Your DBMS should have a data definition and manipulation language, which implements a subset of SQL.Your command parser should accept input from the command line and parse it to check the task to be performed . Each command in the language should be terminated with a semicolon.(i.e. the main unit of execution in your language is a command followed by a semicolon, just like in Java or Sqlplus).

You can assume that a command will never have a newline character in it (the whole command will be written as a single line with a semicolon at the end).Upon parsing a command, you should output an error message if the input is not acceptable in the grammar of your language and do the necessary processing if it is acceptable.

Use an input scanner, such as the Scanner class in Java, to accept input and include a method for parsing the input into individual tokens to determine its compliance with the language rules as well as the corresponding action to be taken by the DBMS.

You can also assume that the reserved keywords will be input in all uppercase, but you can feel free to implement case-insensitive keywords if you prefer.

Detailed description of the structure of commands:
Note that the commands you will implement for this project have a more restricted structure than regular SQL commands (for easy implementation). The command forms specified here are a subset of the forms acceptable by SQL, therefore even if you implement the whole SQL language, it will still cover the command forms in this project. You can also assume that the reserved keywords will be input in all uppercase, but you can feel free to implement case-insensitive keywords if you prefer. Note that you are not asked to implement nested statements in any of the commands.

1. CREATE TABLE:
This command has the form:

CREATE TABLE table_name (attribute_1 attribute1_type CHECK (constraint1), attribute_2 attribute2_type, …, PRIMARY KEY (attribute_1, attribute_2), FOREIGN KEY (attribute_y) REFERENCES table_x (attribute_t), FOREIGN KEY (attribute_w) REFERENCES table_y (attribute_z)…);

The “CREATE TABLE” token is followed by any number of attribute name – attribute type pairs separated by commas. Each attribute name – attribute type pair can optionally be followed by a constraint specified using the keyword “CHECK” followed by a domain constraint.(Note that optional means that the input by the user is optional, not the implementation). This is followed by the token “PRIMARY KEY” and a list of attribute names separated by commas, enclosed in parentheses. Note that the specification of the primary key constraint is mandatory in this project and will always follow the listing of attributes. After the primary key constraint, the command should accept an optional list of foreign key constraints specified with the token “FOREIGN KEY” followed by an attribute name enclosed in parentheses, followed by the keyword “REFERENCES”, a table name and an attribute name enclosed in parentheses. Multiple foreign key constraints are separated by commas.

The output should be “Table created successfully” if table creation succeeds, and a descriptive error message if it fails.

2. DROP TABLE:
This command has the form:

DROP TABLE table_name;

The “DROP TABLE” token is followed by a table name.

The output should be “Table dropped successfully” if table dropping succeeds, and a descriptive error message if it fails.

3. DESCRIBE T_NAME:
This command has the form:

DESCRIBE table_name;

The token “DESCRIBE” is followed by a table name.
The output should be the list of attribute names and types in the table and a list of any constraints (primary key, foreign key, domain), one row for each attribute. If there are any constraints for an attribute, you should print the primary key constraint first, foreign key constraint next and domain constraints last. If the table does not exist, you should print an error message.
An example output is as follows:

```
snum -- int -- primary key -- snum>0
sname -- char(30)
age -- int -- age > 0 AND age < 100
deptid -- int -- foreign key references Department(deptid)
```

4. INSERT:
This command has the form:

```
INSERT INTO table_name VALUES ( val1, val2, … );
```

The “INSERT INTO” token is followed by a table name, followed by the token “VALUES” and a list of values separated by commas enclosed in parentheses. Each value should be either a number (integer or decimal) or a string enclosed in single quotes. Note that you are asked to implement only one form of this command, where the values listed are inserted into the table in the same order that they are specified, i.e. the first value corresponds to the value of the first attribute, the second value corresponds to the value of the second attribute etc. Note that to satisfy this requirement, you should store the ordering of attributes when a table is created.The output should be the message “Tuple inserted successfully” if the insertion succeeds, and a descriptive error message if it fails. Note that you need to check for any constraints specified on the table to make sure they are not violated before proceeding with the insertion. Specifically, you should check that:
a) A record with the same primary key value does not already exist in the database
b) The values inserted comply with any domain constraints defined on the table
c) If any foreign key constraints are defined, there should be matching values in the tables referred to.

5. DELETE:
This command has the form:

```
DELETE FROM table_name WHERE condition_list;
```

The “DELETE FROM” token is followed by a table name, followed by the optional “WHERE” keyword and a condition list. The condition list has the following format:

```
attribute1 operator value1
OR
attribute1 operator value1 AND/OR attribute2 operator value2 AND/OR attribute3
operator value3…
```

The operator can be any of "=", "!=", "<", ">", "<=", ">=".
For simplicity, you can assume that if there are multiple conjunction/disjunction operators in the predicate, they will all be the same operator (i.e. there will not be AND and OR operators mixed in the same condition). Hence, the conditions do not need to be enclosed in parentheses.
The output should be the message "X rows affected", where X is the number of tuples deleted if there are no errors. Otherwise a descriptive error message should be printed.

6. UPDATE:
This command has the form:

UPDATE table_name SET attr1 = val1, attr2 = val2… WHERE condition_list;

The "UPDATE" token is followed by a table name, which is followed by the token "SET" and a list of attribute name=attribute value pairs separated by commas. This is followed by an optional "WHERE" token and a condition list in the same form as the condition list in the DELETE command.
The output should be the message "X rows affected", where X is the number of tuples updated if there are no errors. Otherwise a descriptive error message should be printed.

7. SELECT:
This command has the form:

SELECT attribute_list FROM table_list WHERE condition_list;

The token "SELECT" is followed by an attribute list, followed by the token "FROM" and a table name list. This is followed by an optional "WHERE" keyword and condition list. For simplicity, you are only asked to implement an attribute list consisting of attribute names separated by commas and not using the dot notation, in addition to "*", which stands for all attributes. You can also assume that no attributes of different tables will have the same name. The table list will also be a simple list of table names separated by commas. The condition list has the following format:

attribute1 operator value1
OR
attribute1 operator value1 AND/OR attribute2 operator value2 AND/OR attribute3 operator value3…

The operator can be any of "=", "!=", "<", ">", "<=", ">=".
For simplicity, you can assume that if there are multiple conjunction/disjunction operators in the predicate, they will all be the same operator (i.e. there will not be AND and OR operators mixed in the same condition). Hence, the conditions do not

need to be enclosed in parentheses. The values in the conditions can either be a constant value or the name of another attribute.

An example command is as follows:
SELECT num FROM Student, Enrolled WHERE num = snum AND age > 18;

assuming num and age are attributes of the Student table and snum is an attribute of the Enrolled table.
The output of this command should be the list of matching tuples if there is no error. Otherwise, a descriptive error message should be printed. The first line of the result should be the names of the attributes separated by tab characters (as you would get from Sqlplus). Then you should print the tuples, one line per record, with different attribute values separated by tab characters.

8. HELP TABLES:
This command has the form:

HELP TABLES;

The output should be the list of tables in the database, with one row per table name. If there are no tables in the database, you should print the message "No tables found".

9. HELP CMD:
This command has the form:

HELP command;

The token "HELP" is followed by a command, where command is one of the following:
CREATE TABLE
DROP TABLE
SELECT
INSERT
DELETE
UPDATE
The output should be a short description of the corresponding command and its expected format.

10. QUIT:
This command has the form:

QUIT;

Upon issuance of this command, the program should commit any pending changes to the disk and terminate.

Persistent data management
Your DBMS should persist data across different runs, i.e. if there are any transactions changing the state of the database in one run of your program (i.e. table creation/dropping, record insertions/deletions/updates), the changes should persist after the program is ended. To achieve this, you should commit transactions to the disk by saving the changes in the files that store the data for your DBMS. For simplicity, you can assume that your DBMS will exit properly after each run (i.e. it will not crash), therefore you can keep track of changes to the database in memory during runtime and commit to the disk before exiting the program.
For easy data persistence, it is advisable to store all relation schemas in one file, and have a separate file for the actual data (tuples) in each relation. You can take advantage of object serialization to save data to/retrieve data from a file if Java is your language of implementation. Vector/ArrayList classes in Java also come in handy for keeping a list of tuples.

There are three main classes of operations that would result in changes to the state of the database:
1. Create Table: This operation results in the addition of a relation schema to the list of all schemas in the database. When the user issues this command, you should make sure there is no table with the same name in the database before adding the table to the list of schemas. If a table with the same name exists, the command should result in an error. Note that you also need to take care of any foreign key constraints, making sure that the tables and attributes referred to when creating this table already exist in the list of schemas as well.

2. Drop Table: This operation results in the removal of a relation schema from the list of all schemas in the database. When the user issues this command, you should make sure the specified table already exists in the database. Otherwise the command should result in an error. Upon issuance of this command, the specified table should be removed from the list of schemas and all tuples of the table should be deleted too. Note that you also need to check for foreign key constraints before deleting a table. If any other tables in the database have attributes referring to any attributes of this table, the command should result in an error.

3. Insert/Delete/Update a record: These operations result in changes to the list of records/attributes of records in individual relation files. When inserting a record into the database, you should check that the primary key value of the record to be inserted does not match the primary key value of any existing record in the same table. Note that for simplicity, you do not have to check for referential integrity when deleting records. You can assume that no record with attributes referred to by another record in the database will be deleted.