LAB MANUAL

CSE 4308

DATABASE MANAGEMENT SYSTEMS LAB

Table of Contents

1 SECTION

MySQL Database & User Creation

1.1 Create MySQL databases and users

Once you have MySQL installed on the server(s) that will host your MySQL environment, you need to create a database and additional user accounts. In order to run the following commands, log into the MySQL instance with the MySQL root account.

1.1.1 Create a MySQL database

To create a database called 'strongdm', type the following into the MySQL command line:

```
mysql> CREATE DATABASE strongdm;
```

If the database does not already have a unique name for the MySQL instance, MySQL will issue an error message with error code 1007. Add IF NOT EXISTS to the command to prevent this error with the code below

```
mysql> CREATE DATABASE IF NOT EXISTS strongdm;
```

1.1.2 Delete a MySQL database

```
mysql> DROP DATABASE strongdm;
```

1.1.3 Create a new MySQL user account

MySQL defines users with a username and the hostname or IP address that they're using to access the MySQL instance. To create a new user in MySQL, specify the username, the hostname the user can use to access the database management system, and a secure password:

```
mysql> CREATE USER 'local_user'@'localhost' IDENTIFIED BY 'password'
    ;
```

This command will allow the user with username local_user to access the MySQL instance from the local machine (localhost) and prevent the user from accessing it directly from any other machine. Alternatively, you can use a wildcard character (%) in the host definition to grant access to the MySQL instance for a user:

```
mysql> CREATE USER 'subnet_user'@'10.0.%' IDENTIFIED BY 'password';
```

In the above example, the '10.0.%' specifies that the user can access the MySQL instance from any client that has an IP address beginning with '10.0.'. You may use the wild card at any level of the IP address in the host definition.
To view all users in your MySQL instance, use the SELECT command:

```
mysql> SELECT * FROM mysql.user;
```

1.1.4 Granting a User Permissions

```
GRANT PRIVILEGE ON database.table TO 'username'@'host';

GRANT CREATE, ALTER, DROP, INSERT, UPDATE, DELETE, SELECT,
    REFERENCES, RELOAD on *.* TO 'sammy'@'localhost' WITH GRANT
    OPTION;

REVOKE type_of_permission ON database_name.table_name FROM 'username
    '@'host';

SHOW GRANTS FOR 'username'@'host';

DROP USER 'username'@'localhost';
```

SECTION 2

Data Definition and Data Manipulation

2.1 Data Definition

2.1.1 Creating a Table

The general syntax for creating a table in MySQL is as follows:

```sql
CREATE TABLE table_name (
    attribute1 datatype [NOT NULL],
    attribute2 datatype [NOT NULL],
    ...
);
```

There exist different data types in MySQL. Some of them are as follows:

- `CHAR(n)`: value contains exactly n alpha-numeric characters
- `VARCHAR(n)`: value contains at most n alpha-numeric characters
- `INT`, `DECIMAL`, `FLOAT`, `DOUBLE`: numeric types
- `DATE`, `DATETIME`: used for dates

Assume that you want to create a table named ‘TEST‘ with 3 attributes:

```sql
CREATE TABLE CITIZEN (
    NATIONAL_ID INT NOT NULL,
    NAME VARCHAR(50) NOT NULL,
    BIRTH_DATE DATE
);
```

To create a table with constraints in MySQL:

```
CREATE TABLE table_name (
    attribute1 datatype [NOT NULL],
    attribute2 datatype [NOT NULL],
    ...,
    PRIMARY KEY (primary_attribute1, ...),
    CHECK (condition)
);
```

Primary key is a special column that is able to uniquely identify each record.
For example:

```
CREATE TABLE CITIZEN (
    NATIONAL_ID INT NOT NULL,
    NAME VARCHAR(50) NOT NULL,
    AGE INT,
    COUNTRY VARCHAR(20),
    PRIMARY KEY (NATIONAL_ID),
    CONSTRAINT VALIDITY_CHECK CHECK (AGE > 17 AND COUNTRY = '
   Bangladesh')
);
```

2.1.2 Dropping Tables

To delete the schema in MySQL:

```
DROP TABLE table_name;
```

To delete the table structure with constraints:

```
DROP TABLE table_name CASCADE;
```

2.1.3 Altering Tables

To add a new attribute to the table (can be multiple):

```
ALTER TABLE table_name ADD (attribute_name1 datatype, ...);
```

To delete an attribute from a table (can be multiple):

```
ALTER TABLE table_name DROP COLUMN  (attribute_name1, ...);
```

To modify the data type of an attribute, we need to ensure that the column is empty. Then execute:

```
ALTER TABLE table_name MODIFY attribute_name new_datatype;
```

To rename an attribute:

```
ALTER TABLE table_name RENAME COLUMN old_attribute_name TO
   new_attribute_name;
```

To rename a table:

```
ALTER TABLE table_name RENAME TO new_table_name;
```

Composite Primary key & Foreign key

When a single attribute cannot uniquely identify rows in a table, we use a composite primary key, which consists of multiple columns. Additionally, foreign keys link the values of one table to another. Here's how you declare them in MySQL:

```
CREATE TABLE table_name (
    attribute1 datatype [NOT NULL],
    attribute2 datatype [NOT NULL],
    ...,
    PRIMARY KEY (primary_attribute1, ...),
    FOREIGN KEY (foreign_attribute1) REFERENCES reference_table_name
    [ON DELETE CASCADE | ON DELETE SET NULL | ON DELETE SET DEFAULT]
    [ON UPDATE CASCADE | ON UPDATE SET NULL | ON UPDATE SET DEFAULT]
);
```

Example:

```
CREATE TABLE DEPARTMENT (
    DEPT_NAME VARCHAR(20),
    TITLE VARCHAR(30),
    EST_YEAR CHAR(4),
    PRIMARY KEY (DEPT_NAME)
);

CREATE TABLE COURSE (
    COURSE_ID VARCHAR(8),
    TITLE VARCHAR(30),
    PROGRAM VARCHAR(5),
    DEPT_NAME VARCHAR(20),
    CREDITS INT,
    PRIMARY KEY (COURSE_ID, PROGRAM),
    FOREIGN KEY (DEPT_NAME) REFERENCES DEPARTMENT(DEPT_NAME)
    ON DELETE CASCADE
);
```

You can also copy a table's schema and data:

```
CREATE TABLE new_table_name AS SELECT * FROM old_table_name;
```

Altering Table Constraint

To add or remove constraints:

```
ALTER TABLE table_name ADD CONSTRAINT constraint_name CHECK (
   condition);
ALTER TABLE COURSE ADD CONSTRAINT CK_CREDIT CHECK (CREDIT > 0.75);

ALTER TABLE table_name DROP CONSTRAINT constraint_name;
ALTER TABLE COURSE DROP CONSTRAINT CK_CREDIT;

ALTER TABLE table_name RENAME CONSTRAINT old_constraint_name TO
   new_constraint_name;
ALTER TABLE COURSE RENAME CONSTRAINT CK_CREDIT TO CRDT_CHK;
```

2.2 Data Manipulation

2.2.1 Inserting Records/Rows into a Table

The general format for inserting a new record in MySQL:

```sql
INSERT INTO TABLE_NAME VALUES (...);
```

Positional and named notation examples:

```sql
INSERT INTO CITIZEN VALUES (2015001, 'W', 19, 'Bangladesh');
INSERT INTO CITIZEN (NATIONAL_ID, NAME, AGE, COUNTRY) VALUES
    (2015002, 'X', 23, 'Bangladesh');
```

2.2.2 Retrieval of Information

The basic SQL syntax of a query in MySQL:

```sql
SELECT ATTRIBUTE1, ATTRIBUTE2 FROM TABLE_NAME WHERE <CONDITIONAL
    CLAUSE>;
```

Examples:

```sql
SELECT NATIONAL_ID FROM CITIZEN;
SELECT NATIONAL_ID FROM CITIZEN WHERE ID = 2015001;
SELECT NATIONAL_ID FROM CITIZEN WHERE AGE > 21 AND COUNTRY = '
    Bangladesh';
SELECT NATIONAL_ID FROM CITIZEN WHERE AGE > 21 OR NAME = 'W';
SELECT * FROM CITIZEN;
```

Data Retrieval from Multiple Tables

```sql
SELECT * FROM COURSE, DEPARTMENT;
SELECT * FROM COURSE JOIN DEPARTMENT ON COURSE.DEPT_NAME =
    DEPARTMENT.DEPT_NAME;
```

Update Information of the Table

```sql
UPDATE table_name SET attribute1 = value1, attribute2 = value2 WHERE
     condition;
UPDATE COURSE SET title = 'Database Management Systems', Credits = 3
     WHERE COURSE_ID = 'CSE4307';
```

Delete Information of the Table

```sql
DELETE FROM table_name WHERE condition;
DELETE FROM COURSE WHERE COURSE_ID = 'CSE4307';
```

Be careful with the `UPDATE` and `DELETE` commands without a `WHERE` clause, as they will modify or delete all records!

2.3 Executing SQL Script in MySQL

Suppose, you have written your SQL statements in a file ‘a.sql‘ saved under ‘d:sample‘ directory. To execute that script, you type:

```
source d:/sample/a.sql;
```

3

SECTION

Advanced Data Manipulation

3.1 Distinct

The DISTINCT statement is used to return only distinct (different) values. Inside a table, a column often contains many duplicate values; sometimes you only want to list the different (distinct) values.

```
SELECT DISTINCT attributename FROM tablename;
```

Example:

```
SELECT DISTINCT DEPT_NAME FROM EMPLOYEE;
```

3.2 Range Operator

3.2.1 BETWEEN Operator

The BETWEEN operator selects values within a given range. The values can be numbers, strings or dates. The BETWEEN operator is inclusive: both the begin and end values are included. You must provide the lower value first then the upper value.

```
SELECT attribute_1, .... FROM tablename WHERE attribute_1 BETWEEN
    l_value AND u_value;
```

Example:

```
SELECT NAME, DEPT_NAME, SALARY FROM EMPLOYEE WHERE SALARY BETWEEN
    35000 AND 80000;
```

For excluding a range, use NOT before BETWEEN:

```sql
SELECT NAME, DEPT_NAME, SALARY FROM EMPLOYEE WHERE SALARY NOT
    BETWEEN 35000 AND 80000;
```

3.2.2 IN Operator

The `IN` operator allows you to specify multiple values in a `WHERE` clause. It is a shorthand for multiple `OR` conditions.

```sql
SELECT attribute_1, .... FROM tablename WHERE attribute_1 IN (value1
    , value2, ...);
```

Example:

```sql
SELECT NAME, SALARY FROM EMPLOYEE WHERE DEPT_NAME IN ('DEV', '
    TESTING');
```

3.2.3 Rename

SQL aliases are used to give a table, or a column in a table, a temporary name.

```sql
SELECT ID, NAME, (SALARY * 12) AS Y_SALARY FROM EMPLOYEE;
```

3.3 String Operator

3.3.1 LIKE Operator

The `LIKE` operator is used to match substrings in a query.

```sql
SELECT attribute_1, attribute_2 FROM tablename WHERE attribute_1
    LIKE pattern;
```

Example:

```sql
SELECT NAME FROM EMPLOYEE WHERE NAME LIKE 'A%';
SELECT NAME FROM EMPLOYEE WHERE NAME LIKE 'A___';
```

3.3.2 Concatenation

To concatenate multiple columns or any additional string with any column at the time of retrieving data, SQL supports the string operator (||).

```sql
SELECT 'Employee NAME: ' || NAME FROM EMPLOYEE;
```

3.4 Null Value Handling

Handling null values in SQL operations:

```sql
SELECT NAME, (SALARY + IFNULL(BONUS, 0)) AS TOTAL FROM EMPLOYEE;
```

Now the null values found will be treated as 0, so total salary is now meaningful.

3.5 Set Operator

3.5.1 UNION

The `UNION` operator is used to retrieve all data vertically and keeps the common data only once.

```
SELECT NAME FROM EMPLOYEE WHERE DEPT_NAME ='DEV' UNION SELECT NAME
    FROM EMPLOYEE WHERE DEPT_NAME ='TESTING';
```

3.5.2 INTERSECTION

For retrieving common data from both tables, use the `INTERSECT` operator.

```
SELECT NAME FROM EMPLOYEE WHERE DEPT_NAME ='DEV' INTERSECT SELECT
    NAME FROM EMPLOYEE WHERE DEPT_NAME ='TESTING';
```

3.6 Data Sorting

The `ORDER BY` keyword sorts the result-set in ascending or descending order.

```
SELECT NAME, SALARY FROM EMPLOYEE ORDER BY DEPT, SALARY DESC;
```

3.7 Aggregation Function

SQL aggregate functions include `avg`, `min`, `max`, `sum`, and `count`.

```
SELECT avg(SALARY) FROM EMPLOYEE;
```

3.8 Subquery

SQL subqueries are nested queries.

```
SELECT DEPT FROM EMPLOYEE WHERE SALARY > (SELECT avg(SALARY) FROM
    EMPLOYEE);
```

3.9 Some Date Functions

3.9.1 Current Date

MySQL provides functions like `CURDATE()` and `NOW()` to get the current date:

```
SELECT CURDATE();
SELECT NOW();
```

3.9.2 Date Conversion

MySQL uses `STR_TO_DATE()` for converting strings to dates:

```
SELECT STR_TO_DATE('20 APR 2020', '%d %b %Y');
```

3.9.3 Format Date

In MySQL, `DATE_FORMAT()` is used to format dates:

```sql
SELECT DATE_FORMAT(NOW(), '%d-%m-%Y');
```

3.9.4 Extraction

Extract year, month, or day using `EXTRACT()`:

```sql
SELECT EXTRACT(YEAR FROM STR_TO_DATE('29-Apr-2020 05:30:20', '%d-%b
    -%Y %H:%i:%s'));
```

3.9.5 Last Day

To get the last day of the month, use `LAST_DAY()`:

```sql
SELECT LAST_DAY(NOW());
```

3.9.6 Next Day

MySQL uses `ADDDATE()` to find the date after a specific day:

```sql
SELECT ADDDATE(NOW(), INTERVAL 1 DAY);
```

3.9.7 Months Between

To calculate months between two dates, MySQL uses `TIMESTAMPDIFF()`:

```sql
SELECT TIMESTAMPDIFF(MONTH, '2011-04-02', NOW());
```

3.9.8 Add Months

`ADDDATE()` is also used to add months to a date:

```sql
SELECT ADDDATE(NOW(), INTERVAL 2 MONTH);
```

3.9.9 Adding Days

Adding days directly to dates:

```sql
SELECT NOW() + INTERVAL 10 DAY;
```

3.10 Some String Functions

3.10.1 Length

To get the length of a string:

```sql
SELECT LENGTH('HELLO');
```

3.10.2 Lower

Convert to lowercase:

```sql
SELECT LOWER('Hello');
```

3.10.3 Upper

Convert to uppercase:

```sql
SELECT UPPER('Hello');
```

3.10.4 Initcap

Capitalize each word:

```sql
SELECT INITCAP('HELLO');
```

3.10.5 Trim

Trimming spaces or specific characters:

```sql
SELECT TRIM('  Removing Leading White Spaces  ');
SELECT TRIM(LEADING '6' FROM '660123');
SELECT TRIM(TRAILING '5' FROM '123455');
```

3.10.6 LPAD and RPAD

Padding a string on the left or right:

```sql
SELECT LPAD('Hello', 10, '+');
SELECT RPAD('Hello', 10, '+');
```

3.10.7 Replace

Replacing parts of a string:

```sql
SELECT REPLACE('JACK and JUE', 'J', 'BL');
```

4

SECTION

Views and Roles

4.1 Views

A **View** in MySQL is a virtual table based on the result set of a query. It can be used to simplify complex queries, provide data security by limiting access to specific data, and present a specific view of the database for different users.
Generalized command:

```sql
CREATE [OR REPLACE] VIEW view_name AS
SELECT columns
FROM tables
[WHERE conditions];
```

4.1.1 Example of a View

Suppose we have a table named `employees` in the database:

```sql
CREATE TABLE employees (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(50),
    position VARCHAR(50),
    salary DECIMAL(10, 2)
);
```

We can create a view that only shows employee names and positions, hiding their salary:

```sql
CREATE VIEW employee_overview AS
SELECT name, position
FROM employees;
```

Now, users who query the `employee_overview` view will only see the names and positions of employees:

```sql
SELECT * FROM employee_overview;
```

This provides a layer of abstraction and security, as the salary details are not exposed.

Unlike a table, a view does not store any data. To be precise, a view only behaves like a table. And it is just a named query stored in the database. This is why sometimes a view is referred to as a named query.

4.1.2 Simple vs Complex Views

The following table summarizes the differences between a Simple View and a Complex View in MySQL.

Simple View	**Complex View**
Contains only one single base table or is created from only one table.	Contains more than one base table or is created from more than one table.
Simple view does not contain aggregated functions, `GROUP BY`, `DISTINCT`, pseudo-columns like `ROWNUM`, or columns defined by expressions.	It can contain aggregated functions, `GROUP BY`, `DISTINCT`, pseudo-columns like `ROWNUM`, and columns defined by expressions.
Does not include `NOT NULL` columns from base tables.	`NOT NULL` columns that are not selected by a simple view can be included in complex views.

4.1.3 Working with Views

In the case of complex views, views cannot be updated, and they cannot be used to modify the parent table they are generated from. However, simple views can be used to update or modify the parent table. Any change you perform in your simple view will be reflected in the parent table.
Assume `MOVIE_SUMMARY` is a simple view, we can modify it. For example:

```sql
INSERT INTO MOVIE_SUMMARY
VALUES ('935', 'Emily the Criminal', 'English');
```

To update data in the simple view:

```sql
UPDATE MOVIE_SUMMARY
SET MOV_ID = '934'
WHERE MOV_ID = '935';
```

To delete a record from the simple view:

```sql
DELETE FROM MOVIE_SUMMARY
WHERE MOV_TITLE = 'Emily the Criminal';
```

One of the major use cases for views (complex view) is for simplifying data retrieval. For example, the following query can be used to determine the genres where actors are preferred to actresses:

```
SELECT GEN_TITLE
FROM GENRES G
WHERE ( SELECT COUNT (*)
FROM ACTOR NATURAL JOIN CASTS NATURAL JOIN MTYPE
NATURAL JOIN
GENRES
WHERE ACT_GENDER =    M    AND GEN_TITLE = G.GEN_TITLE
GROUP BY GEN_TITLE ) > ( SELECT COUNT (*)
FROM ACTOR NATURAL JOIN CASTS NATURAL JOIN MTYPE
NATURAL JOIN
GENRES
WHERE ACT_GENDER =    F    AND GEN_TITLE = G.GEN_TITLE
GROUP BY GEN_TITLE );
```

However, by adding the line:

```
CREATE OR REPLACE VIEW SEXIST_GENRES AS
```

before the query, we can create a view which can then be used to simplify other queries. Another benefit of using views can be providing an additional security layer. They help us to hide certain columns and rows from the underlying tables and expose only needed data to the appropriate users. For example, a STUDENT table may store the student ID, name, department, CGPA of different students. But we might want to share only the student ID and department of the student with the instructors.

4.1.4 Drop Views

```
DROP VIEW view_name;
```

4.2 Roles

A **Role** in MySQL is a named collection of privileges that can be granted to users. This allows for easier management of user privileges by grouping them into roles.

4.2.1 Example of a Role

Let's create a role called `manager_role` with specific privileges:

```
CREATE ROLE manager_role;
GRANT SELECT, INSERT, UPDATE ON employees TO manager_role;
```

Now, we can assign this role to a user, such as `user1`:

```
GRANT manager_role TO 'user1'@'localhost';
```

This simplifies privilege management since you can now modify the role without affecting individual user permissions.

For example, we can create a role for people who want to watch movies and rating of the movies, a role for reviewers who can watch movies and rating also give ratings, and finally, an admin who can add, remove or update movie

```sql
-- Create the Viewer role
CREATE ROLE Viewer;

-- Grant SELECT privilege to the Viewer role on MOVIE and RATING
   tables
GRANT SELECT ON [USERNAME].MOVIE TO Viewer;
GRANT SELECT ON [USERNAME].RATING TO Viewer;

-- Grant SELECT privilege to the Viewer role on the MOVIE_SUMMARY
   view
GRANT SELECT ON [USERNAME].MOVIE_SUMMARY TO Viewer;

-- Revoke privilege from the Viewer role
REVOKE SELECT ON [USERNAME].MOVIE_SUMMARY FROM Viewer;

-- Create another role called Reviewer
CREATE ROLE Reviewer;

-- Grant the Viewer role to the Reviewer role
GRANT Viewer TO Reviewer;

-- Grant INSERT privilege to the Reviewer role on RATING table
GRANT INSERT ON [USERNAME].RATING TO Reviewer;

-- Create Admin role with multiple privileges on MOVIE table
CREATE ROLE Admin;
GRANT SELECT, INSERT, UPDATE, DELETE ON [USERNAME].MOVIE TO Admin;

-- Alternatively, grant all privileges on MOVIE table to Admin
GRANT ALL PRIVILEGES ON [USERNAME].MOVIE TO Admin;

-- Grant Admin the ability to grant privileges to others
GRANT ALL PRIVILEGES ON [USERNAME].MOVIE TO Admin WITH GRANT OPTION;

-- Create users and grant privileges
DROP USER IF EXISTS 'V_103';
DROP USER IF EXISTS 'R_432';
DROP USER IF EXISTS 'Admin_01';

CREATE USER 'V_103' IDENTIFIED BY 'test123';
GRANT ALL PRIVILEGES ON *.* TO 'V_103';

CREATE USER 'R_432' IDENTIFIED BY 'test123';
GRANT ALL PRIVILEGES ON *.* TO 'R_432';

CREATE USER 'Admin_01' IDENTIFIED BY 'test123';
GRANT ALL PRIVILEGES ON *.* TO 'Admin_01';

```

```sql
-- Assign each user a role
GRANT Viewer TO 'V_103';
GRANT Reviewer TO 'R_432';
GRANT Admin TO 'Admin_01';

-- Assign multiple roles to Admin_01
GRANT Viewer, Reviewer TO 'Admin_01';


INSERT INTO dbms.Rating VALUES (913, 9010, 5);
```

4.3 Combining Views and Roles

By combining Views and Roles, you can create a secure and manageable database environment. For example, you can restrict a `manager` to only access specific views and perform limited operations on certain tables.