National University of Sciences and Technology (NUST)
School of Electrical Engineering and Computer Science

High Impact Skills Development Program in Artificial Intelligence, Data Science, and Blockchain

Lab 02: SQL Operators, DDL, and Constraints

Instructor: Mr. Bilal Ali

Introduction

Structured Query Language (SQL) was developed at IBM San Jose Research Laboratory as a part of System R project. It is a declarative query language for querying a relational database. It also includes features for defining the structure of the data, for inserting and modifying data in the database, and for specifying security constraints. It is relational complete (it supports all six core relational algebra operations). SQL commands can be classified into three groups DDL, DML & DCL.

Objectives

After performing this lab students should be able to:

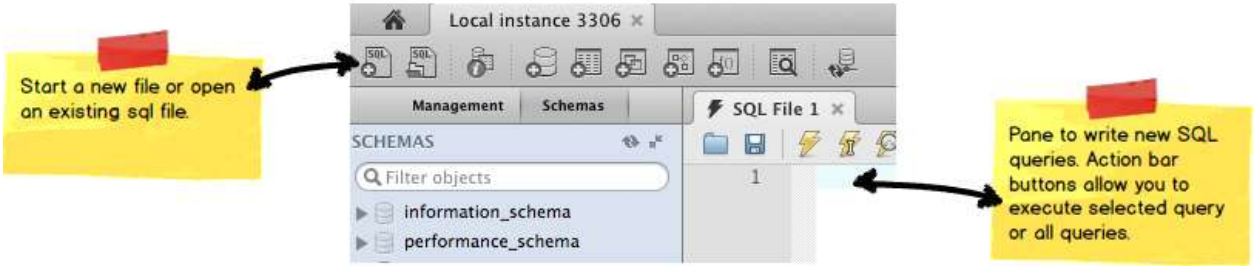
1. Design SQL queries to retrieve data using SELECT clause and various associated operators.
2. Design SQL queries with FROM & WHERE clause.
3. Execute SQL queries over MySQL using MySQL Workbench.
4. Learn about string function and execute them in a query.

Tools/Software Requirement

- MySQL Community Server
- MySQL Workbench
- Sakila Database

Description

1. Open MySQL Workbench and open the default connection instance.
2. A new query window would open from where you can write and execute queries.



3. You can save the query file and can also add comments using # symbol.
4. On executing queries, results are displayed in the lower part of the screen.
5. Error or success messages are displayed in action output pane at the bottom.
6. Try running few SQL queries modeled during the lectures to get it going.
7. Continue playing with the Workbench and SQL queries till you are comfortable with the querying mechanism and have learnt the shortcuts to execute queries.

1. SQL Basic Query Structure

The SELECT clause

The most common use of SQL commands is the selection of data from the tables located in a database. This can be achieved through the SELECT command. We need to SELECT information FROM a table. Hence we have the most basic SQL query structure comprising of:

- SELECT
- FROM
- WHERE

The syntax for the SELECT clause is as follows:

```
SELECT "column_name(s)" FROM "table_name(s)";
```

Always specify the name of the database in which a table/relation is present through which data is to be retrieved.

e.g. ***Select column_name***
From Sakila.table_name

There are three ways we can retrieve data from a table:

- Retrieve one column
- Retrieve multiple columns
- Retrieve all columns (Use *)

The select clause can contain arithmetic expressions involving the operation, +, –, *, and /, and operating on constants or attributes of records (tuples).

The FROM clause

The FROM clause can allow to select attributes from a single table or multiple tables. When multiple tables are applied, it combines the records from the two or more tables listed and presents every possible combination of the listed attributes in the SELECT clause.
This is only very useful once some filtering condition is applied. This is achieved through using the WHERE clause.

The WHERE clause

We can use the WHERE clause to filter the result set based on certain conditions. The syntax for using WHERE in the SELECT statement is as follows:

```
SELECT "column_name(s)"
FROM "table_name(s)"
WHERE "condition";
```

"Condition" can include a single comparison clause (called simple condition) or multiple comparison clauses combined together using AND or OR operators (compound condition). Conditions can include other operators like IN, BETWEEN, DISTINCT etc shown in Table 1:

Operator	Description	Example
=	Equal to	Author = 'Alcott'
<>	Not equal to (most DBMS also accept != instead of <>)	Dept <> 'Sales'
>	Greater than	Hire_Date > '2012-01-31'
<	Less than	Bonus < 50000.00
>=	Greater than or equal	Dependants >= 2
<=	Less than or equal	Rate <= 0.05

BETWEEN	Between an inclusive range	Cost BETWEEN 100.00 AND 500.00
LIKE	Match a character pattern	First_Name LIKE 'Will%'

Table 1: List of Basic Operators available in SQL

2. Ordering data

The order of rows returned in a query result is undefined. The ORDER BY clause can be used to sort the rows. This clause comes last in the SELECT statement. ASC at the end of the ORDER BY clause specifies ascending order whereas DESC specifies descending order. ASC is the default order. The syntax for an ORDER BY statement is as follows:

```
SELECT "column_name"
FROM "table_name"
[WHERE "condition"]
ORDER BY "column_name" [ASC, DESC];
```

3. The Wildcard operators

There are times when we want to match on a string pattern. To do that, we will need to employ the concept of a wildcard. In SQL, there are two wildcards:

- % (percent sign) represents zero, one, or more characters.
- _ (underscore) represents exactly one character.

Wildcards are used with the LIKE keyword in SQL.
Below are some wildcard examples:

- 'A_Z': All string that starts with 'A', another character, and end with 'Z'. For example, 'ABZ' and 'A2Z' would both satisfy the condition, while 'AKKZ' would not (because there are two characters between A and Z instead of one).
- 'ABC%': All strings that start with 'ABC'. For example, 'ABCD' and 'ABCABC' would both satisfy the condition.
- '%XYZ': All strings that end with 'XYZ'. For example, 'WXYZ' and 'ZZXYZ' would both satisfy the condition.
- '%AN%': All strings that contain the pattern 'AN' anywhere. For example, 'LOS ANGELES' and 'SAN FRANCISCO' would both satisfy the condition.
- '_AN%': All strings that contain a character, then 'AN', followed by anything else. For example, 'SAN FRANCISCO' would satisfy the condition, while 'LOS ANGELES' would not satisfy the condition.

4. ORDER BY clause

The SQL ORDER BY clause allows you to sort the records in your result set. The SQL ORDER BY clause can only be used in **SQL SELECT statements**.

The syntax for the SQL ORDER BY clause is:

```
SELECT columns
FROM tables
WHERE predicates
ORDER BY column ASC/DESC;
```

The SQL ORDER BY clause sorts the result set based on the columns specified. If the ASC or DESC value is omitted, it is sorted by ASC.

ASC indicates ascending order. (default)
DESC indicates descending order.

Example 1:

```
SELECT supplier_city
FROM suppliers
WHERE supplier_name = 'IBM'
ORDER BY supplier_city;
```

Example 2:

When sorting your result set in descending order, you use the DESC attribute in your ORDER BY clause as follows:

```
SELECT supplier_city
FROM suppliers
WHERE supplier_name = 'IBM'
ORDER BY supplier_city DESC;
```

This SQL ORDER BY example would return all records sorted by the supplier_city field in descending order.

SQL ORDER BY - Using both ASC and DESC attributes together

When sorting your result set using the SQL ORDER BY clause, you can use the ASC and DESC attributes in a single **SQL SELECT statement**.

For example:

```
SELECT supplier_city, supplier_state
FROM suppliers
WHERE supplier_name = 'IBM'
ORDER BY supplier_city DESC, supplier_state ASC;
```

This SQL ORDER BY would return all records sorted by the supplier_city field in descending order, with a secondary sort by supplier_state in ascending order.

SQL Operator Precedence

1. Parentheses - if you group conditional SQL statements together by parentheses, SQL Server evaluates the contents of these first.
2. Arithmetic - multiplication (using the operators *, /, or %)
3. Arithmetic - addition (using the operators + or -)
4. Other - String Concatenate (+)
5. Logical - NOT
6. Logical - AND
7. Logical - OR

Note: (The order of evaluation at the same precedence level is from left to right.)

1. Execute the following queries using the Sakila schema.

    Select customer_id, 100*amount-10 from payment;

    Select customer_id, 100*(amount-10) from payment;

Logical - AND

The AND condition allows you to create an SQL statement based on 2 or more conditions being met. It can be used in any valid SQL statement - select, insert, update, or delete.

The syntax for the AND condition is:

```
SELECT columns
FROM tables
WHERE column1 = 'value1'
AND column2 = 'value2';
```

The AND condition requires that each condition be must be met for the record to be included in the result set. In this case, column1 has to equal 'value1' and column2 has to equal 'value2'.

Example #1:

The first example we'll look at involves a very simple example using the AND condition.

```
SELECT *
FROM suppliers
WHERE city = 'New York'
and type = 'PC Manufacturer';
```

This would return all suppliers that reside in New York and are PC Manufacturers. Because the * is used in the select, all fields from the supplier table would appear in the result set.

Logical - OR

The OR condition allows you to create an SQL statement where records are returned when any one of the conditions are met. It can be used in any valid SQL statement - select, insert, update, or delete.

The syntax for the OR condition is:

```
SELECT columns
FROM tables
WHERE column1 = 'value1'
or column2 = 'value2';
```

The OR condition requires that any of the conditions be must be met for the record to be included in the result set. In this case, column1 has to equal 'value1' OR column2 has to equal 'value2'.

Example #1:

The first example we'll look at involves a very simple example using the OR condition.

```
SELECT * FROM suppliers WHERE city = 'New York' or city = 'Newark';
```

This would return all suppliers that reside in either New York or Newark. Because the * is used in the select, all fields from the suppliers' table would appear in the result set.

Example #2:

The next example takes a look at three conditions. If any of these conditions are met, the record will be included in the result set.

```
SELECT supplier_id
FROM suppliers
WHERE name = 'IBM'
or name = 'Hewlett Packard'
or name = 'Gateway';
```

This SQL statement would return all supplier_id values where the supplier's name is either IBM, Hewlett Packard, or Gateway.

Logical Operator Precedence

Logical operators are executed in the following specified order.

1. Logical - NOT
2. Logical - AND
3. Logical - OR

Combination of And & Or

The AND and OR conditions can be combined in a single SQL statement. It can be used in any valid SQL statement - select, insert, update, or delete.

When combining these conditions, it is important to use brackets so that the database knows what order to evaluate each condition.

Example #1:

The first example that we'll take a look at an example that combines the AND and OR conditions.

```
SELECT *
FROM suppliers
WHERE (city = 'New York' and name = 'IBM')
or (city = 'Newark');
```

This would return all suppliers that reside in New York whose name is IBM and all suppliers that reside in Newark. The brackets determine what order the AND and OR conditions are evaluated in.

Example #2:

The next example takes a look at a more complex statement.

```
SELECT supplier_id
FROM suppliers
WHERE (name = 'IBM')
or (name = 'Hewlett Packard' and city = 'Atlantic City')
or (name = 'Gateway' and status = 'Active' and city = 'Burma');
```

This SQL statement would return all supplier_id values where the supplier's name is IBM or the name is Hewlett Packard and the city is Atlantic City or the name is Gateway, the status is Active, and the city is Burma.

Single Row Functions

What is a function?

A function is similar to an operator in operation. A function is a name that performs a specific task. A function may or may not take values (arguments) but it always returns a value as the result. If function takes values then these values are to be given within parentheses after the function name. The following is the general format of a function.

function [(argument-1, argument-2,...)]

If the function doesn't take any value then function name can be used alone and even parentheses are not required.

Single-row functions return a single result row for every row of a queried table or view. These functions can appear in select lists, WHERE clauses, START WITH and CONNECT BY clauses, and HAVING clauses.

Arithmetic functions take numeric data; date functions take date type data and string functions take strings. Conversion functions are used to convert the given value from one type to another. Miscellaneous functions perform operations on any type of data. Group functions are used to perform operations on the groups created by GROUP BY clause.

Character Functions

Character functions operate on values of datatype CHAR or VARCHAR.

LOWER

Returns a given string in lower case.

```
select LOWER(first_name)
```

from actor;

UPPER

Returns a given string in UPPER case.

select UPPER(first_name)

from actor;

LENGTH

Returns the length of a given string.

select length(first_name)

from actor;

CONCAT(str1,str2,...)

Returns the string that results from concatenating the arguments. May have one or more arguments.

```
SELECT CONCAT('My', 'S', 'QL');
+-------------------------------------------------------+
| CONCAT('My', 'S', 'QL')                               |
+-------------------------------------------------------+
MySQL
```

CONCAT_WS(separator,str1,str2,...)

CONCAT_WS() stands for Concatenate With Separator and is a special form of CONCAT(). The first argument is the separator for the rest of the arguments. The separator is added between the strings to be concatenated. The separator can be a string, as can the rest of the arguments. If the separator is NULL, the result is NULL.

```
SELECT CONCAT_WS(',','First name','Last Name' );
+-------------------------------------------------------+
| CONCAT_WS(',','First name','Last Name' )              |
+-------------------------------------------------------+
| First name, Last Name                                 |
```

LPAD(str,len,padstr)

Returns the string str, left-padded with the string padstr to a length of len characters. If str is longer than len, the return value is shortened to len characters.

```
SELECT LPAD('hi',4,'??');
```

```
+-------------------------------------------------------+
| LPAD('hi',4,'??')                                     |
+-------------------------------------------------------+
| ??hi                                                  |
+-------------------------------------------------------+
```

RPAD(str,len,padstr)

Returns the string str, right-padded with the string padstr to a length of len characters. If str is longer than len, the return value is shortened to len characters.

```
SELECT RPAD('hi',5,'?');
+-------------------------------------------------------+
| RPAD('hi',5,'?')                                      |
+-------------------------------------------------------+
| hi???                                                 |
+-------------------------------------------------------+
```

LTRIM(str)

Returns the string str with leading space characters removed.

```
SELECT LTRIM('  lecture');
+-------------------------------------------------------+
| LTRIM('  lecture')                                    |
+-------------------------------------------------------+
| lecture                                               |
+-------------------------------------------------------+
```

REPEAT(str,count)

Returns a string consisting of the string str repeated count times. If count is less than 1, returns an empty string. Returns NULL if str or count are NULL.

```
SELECT REPEAT('MySQL', 3);
+-------------------------------------------------------+
| REPEAT('MySQL', 3)                                    |
+-------------------------------------------------------+
| MySQLMySQLMySQL                                       |
+-------------------------------------------------------+
```

RTRIM(str)

Returns the string str with trailing space characters removed.

```
SELECT RTRIM('barbar   ');
+-------------------------------------------------------+
| RTRIM('barbar   ')                                    |
+-------------------------------------------------------+
| barbar                                                |
+-------------------------------------------------------+
```

SUBSTRING(str,pos)

SUBSTRING(str FROM pos)

SUBSTRING(str,pos,len)

SUBSTRING(str FROM pos FOR len)

The forms without a len argument return a substring from string str starting at position pos. The forms with a len argument return a substring len characters long from string str, starting at position pos. The forms that use FROM are standard SQL syntax. It is also possible to use a negative value for pos. In this case, the beginning of the substring is pos characters from the end of the string, rather than the beginning. A negative value may be used for pos in any of the forms of this function.

```
SELECT SUBSTRING('Quadratically',5);
+-------------------------------------------------------+
| SSUBSTRING('Quadratically',5)                         |
+-------------------------------------------------------+
| ratically                                             |
+-------------------------------------------------------+
> SELECT SUBSTRING('foobarbar' FROM 4);
+-------------------------------------------------------+
| SUBSTRING('foobarbar' FROM 4)                         |
+-------------------------------------------------------+
| barbar                                                |
+-------------------------------------------------------+
> SELECT SUBSTRING('Quadratically',5,6);
+-------------------------------------------------------+
| SUBSTRING('Quadratically',5,6)                        |
```

```
+-----------------------------------------------------+
| ratica                                              |
+-----------------------------------------------------+
```

SUBSTRING_INDEX(str,delim,count)

Returns the substring from string str before count occurrences of the delimiter delim. If the count is positive, everything to the left of the final delimiter (counting from the left) is returned. If count is negative, everything to the right of the final delimiter (counting from the right) is returned. SUBSTRING_INDEX() performs a case-sensitive match when searching for delim.

```
SELECT SUBSTRING_INDEX('www.mysql.com', '.', 2);
+-----------------------------------------------------+
| SUBSTRING_INDEX('www.mysql.com', '.', 2)            |
+-----------------------------------------------------+
| www.mysql                                           |
+-----------------------------------------------------+
```

Arithmetic Operators:

Arithmetic operators and functions are available in MYSQL which are used for arithmetic operations. You can explore them online.

Lab Task

Using Sakila Database

Formulate SQL queries for the following needs and execute them on the Sakila database.

1. Write a query to Select column “actor_id” from the table “actor” which already exists in the “Sakila” database where the “actor_id=58”. (Observe the output when you execute the queries.)
2. Write a query to retrieve the names of movies starting with P.
3. Write a query to retrieve movies that were released in the year 2006.
4. What password did the DBA assign to the user „MIKE“?
5. Write a query to retrieve data of all actors whose first names are not ending with T.
6. Select the names of actors whose IDs are between 50 and 150, or those whose last name starts with A.
7. Select the names of actors whose IDs are between 50 and 150, or those whose last name starts with A.
8. Write a query to display the names of customers in the following format.

SMITH, M
JOHNSON, P
WILLIAMS, L
JONES, B
BROWN, E
DAVIS, J
MILLER, M
WILSON, S
MOORE, M
TAYLOR, D

9. Retrieve the information showing the details of each of the Films released until now in the format: ***Film Academy Dinosaur was released in the year 2006***.
10. Display the usernames and address_ids of customers in the Sakila database.
11. Write two queries using the Substring functions on sakila database.