Enterprise Java with Spring
Spring Data
Lab 1

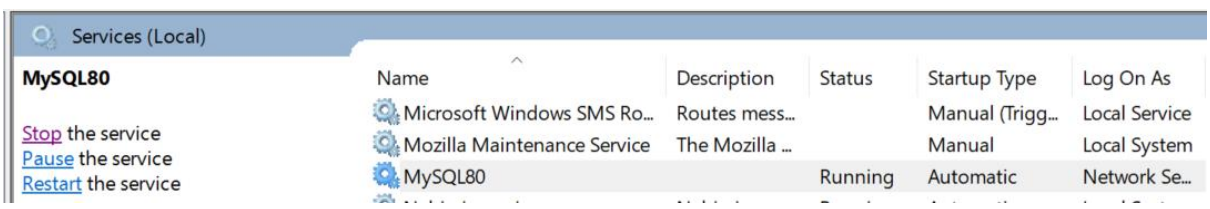
1 Lab setup

Make sure you have the following items installed

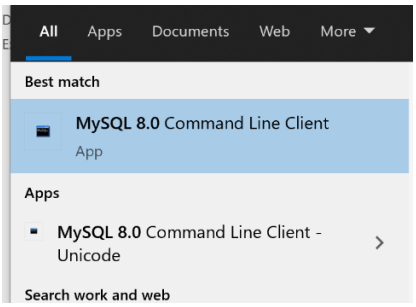
- MySQL 8.xx
- A suitable text editor (Notepad ++)

2 Working with MySQL via the command line

Check that the MySQL server is up and running. This should have already started up as a Windows service by default when your OS boots up. Open the Services utility to check and start the server if it is not already running:



Start up the MySQL command line client from Windows search:



You will be prompted for the admin (root) password. Type this in and you will be presented with the MySQL shell prompt where you can type in SQL statements.

An alternative way to connect to the MySQL server is to open a normal command prompt and type:

```
mysql -u root -p
```

You will again be prompted for the admin (root) password.

To see the list of databases currently available, type:

```
SHOW DATABASES;
```

On a new MySQL installation, you will see some default system databases such as

- `mysql`: Contains essential system tables that store information required by the MySQL server, such as user accounts and privileges.
- `information_schema`: Provides access to database metadata, offering details about database objects like tables, columns, and procedures.
- `performance_schema`: A feature for monitoring MySQL Server execution at a low level, enabling performance analysis and tuning.
- `sys`: A set of objects that helps database administrators and developers interpret data collected by the Performance Schema, simplifying performance and health monitoring tasks.

In addition, there may be also some databases with sample data for practicing SQL queries such as:

- `sakila` - models a DVD rental store, encompassing tables for films, actors, customers, and more
- `world` - contains information about countries, cities, and languages

Create a new database with this command:

```
CREATE DATABASE workshopdb;
```

Check that you have created it properly by listing the databases again with:

```
SHOW DATABASES;
```

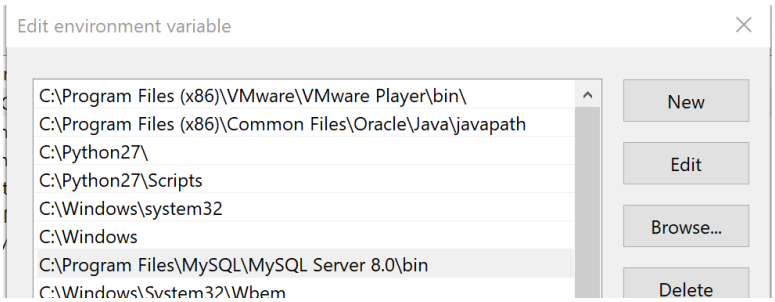
Select the database for use with:

```
USE workshopdb;
```

3 Importing a table into a database

First, add the `bin` subdirectory of your MySQL root installation directory to your system path. Browse through your C:\Program Files to locate this root installation directory: typically it will be something like `C:\Program Files\MySQL\MySQL Server 8.0`

Go to Control Panel -> System and Security -> System -> Advanced System Settings
Select Environment Variables.
In the System Variables box, select the Path system variable and select Edit.
Click New and add in this directory:
`C:\Program Files\MySQL\MySQL Server 8.0\bin`



The table to be imported is in the `sqltables` folder of your `labcode` folder. Open a command prompt in this folder (or navigate to it), then type:

```
mysql -u root -p workshopdb < devtable.sql
```

You will again be prompted for the admin (root) password.

Back in the MySQL command line client, verify that the table has been created with:

```
SHOW TABLES;
```

Check the contents of the table with:

```
SELECT * FROM developers;
```

Notice that the married column has values of either `0x00` or `0x01` . This is because the column is created using a BIT data type to represent a Boolean value (as MySQL does not have a built-in Boolean type).

To see the structure of the table:

```
DESCRIBE developers;
```

The `varchar` type is used to represent strings, and we can see that the `id` column/field has been set to be the `primary` key. We are using a single string to store multiple languages separated by a comma as mySQL does not provide native support for arrays.

4 Using SELECT for queries

We can view the contents of specific columns (rather than all the columns) in the tables by specifying the column names:

```
SELECT name, age FROM developers;

SELECT name, languages, married FROM developers;
```

To limit the display of rows to the first 10 rows:

```
SELECT * FROM developers LIMIT 10;
```

5 Sorting queries with ORDER BY

When you use the SELECT statement to query data from a table, the rows in the result set returned are not sorted in any order. The ORDER BY clause allows you to:

- Sort a result set by a single column or multiple columns.
- Sort a result set by different columns in ascending or descending order.

SELECT *columnList* FROM *tblname* ORDER BY *column1* [ASC|DESC], *column2* [ASC|DESC],... ;

By default, the ORDER BY clause sorts the result set in ascending order if you don't specify ASC or DESC explicitly.

To sort by ascending order of age:

```
SELECT * FROM developers ORDER BY age;
```

To sort by descending order of age:

```
SELECT * FROM developers ORDER BY age desc;
```

To sort by normal alphabetical order on name:

```
SELECT * FROM developers ORDER BY name;
```

We can also perform a secondary sort. For e.g. for the case of developers with identical names, we can choose to sort them on their age by descending order:

```
SELECT name, age FROM developers ORDER BY name, age desc;
```

Similarly for the case of developers with identical ages, we can choose to sort them on their names in ascending order:

```
SELECT age, name FROM developers ORDER BY age desc, name;
```

Since rows are placed in the table in accordance to their id, we can use the id to list the last 10 rows in the table with:

```
SELECT * FROM developers ORDER BY id desc limit 10;
```

6 Filtering queries with WHERE

The WHERE clause is used in conjunction with the SELECT, INSERT, UPDATE, and DELETE statements to specify a subset of rows from a table that meet a particular filtering condition, in order to perform some action on them (retrieve, update, delete, etc). This can be used in combination with other clauses (such as ORDER BY and LIMIT) and with standard relational and logical operators.

To view developers who are older than 40:

```
SELECT name, age FROM developers WHERE age > 40;
```

To view developers who are 30 years old or younger:

```
SELECT name, age FROM developers WHERE age <= 30;
```

To view developers who can code in Python, we need to use the INSTR string function available in MySQL:

```
SELECT name, languages FROM developers WHERE
INSTR(languages,"python") > 0;
```

To view developers who are married, there are several ways to do this as this value is represented as a bit(1) data type in MySQL:

```
SELECT name, married FROM developers WHERE married IS TRUE;
SELECT name, married FROM developers WHERE BIN(married) = 1;
```

Similarly to view developers who are single:

```
SELECT name, married FROM developers WHERE married IS FALSE;
SELECT name, married FROM developers WHERE BIN(married) = 0;
```

We can combine multiple conditions on the filtering using the logical operators AND, OR and NOT:

To view developers who are older than 25 or are single:

```
SELECT name, age, married FROM developers WHERE age > 25 OR married
IS FALSE;
```

To view developers who are older than 40 and are married:

```
SELECT name, age, married FROM developers WHERE age > 40 AND married
IS TRUE;
```

To view developers who are 30 or younger and can code in Java:

```
SELECT name, age, languages FROM developers WHERE age <= 30 AND
INSTR(languages,"java") > 0;
```

To view developers who are married and cannot code in C++:

```
SELECT name, married, languages FROM developers WHERE married IS TRUE
AND INSTR(languages,"C++") < 1;
```

We can use the CASE expression as a form of conditional logic:

CASE value
WHEN *compare_value_1* THEN *result_1*
WHEN *compare_value_2* THEN *result_2*
……
WHEN *compare_value_n* THEN *result_n*
ELSE result END

For e.g. to rename the married column and its binary values to a more readable form:

```
SELECT id, name, age, CASE married
WHEN '0x00' THEN 'Single'
ELSE 'Married'
END
AS maritalStatus FROM developers;
```

Let's say we want to categorize developers into 3 categories based on the languages they can code in:

- Java: Awesome developer
- Python: Normal developer
- Any other language: Crap developer

To provide a listing for these 3 categories:

```
SELECT id, name, languages, CASE
WHEN INSTR(languages,"java") > 0 THEN 'Awesome developer'
WHEN INSTR(languages,"python") > 0 THEN 'Normal developer'
ELSE 'Crap developer'
END
AS STATUS FROM developers;
```

We can combine filtering with sorting, for e.g.

To view developers who are single sorted on descending order of their age:

```
SELECT name, married, age FROM developers WHERE married IS FALSE ORDER
BY age desc;
```

To view developers who are older than 40 and are married sorted on alphabetical order of their names:

```
SELECT name, age, married FROM developers WHERE age > 40 AND married
IS TRUE ORDER BY name;
```

7 Aggregate functions

Aggregate functions are performed on all items of a specified column or expression involving columns, returning a single value.

DISTINCT is used in a query to eliminate duplicate values in a result set. The keyword is always placed immediately after the SELECT keyword and ensures that the expression following it (which should contain at least one column) will return a result set in which all values are unique.

There are many developers that have identical names. To see the list of all unique names:

```
SELECT DISTINCT name from developers;
```

The SUM, AVG, MIN, MAX or COUNT function performs their respective functions on a list of numerical values or expressions involving columns.

To see the combined ages of all the developers:

```
SELECT SUM(age) FROM developers;
```

To see the average age of all the developers:

```
SELECT AVG(age) FROM developers;
```

To get the oldest age among all the developers:

```
SELECT MAX(age) FROM developers;
```

To get the youngest age among all the developers:

```
SELECT MIN(age) FROM developers;
```

Any of these functions can be combined with a filtering using WHERE.

For e.g. to get the age of the oldest developer who is single:

```
SELECT MAX(age) FROM developers WHERE married is false;
```

To get the age of the youngest developer who can code in Java:

```
SELECT MIN(age) FROM developers WHERE INSTR(languages,"java") > 0;
```

The COUNT function returns the number of rows in a result set, either all of them or those that match a particular condition.

To get the total number of rows in the table:

```
SELECT COUNT(*) AS 'Total Developers' FROM developers;
```

To get the total number of married developers:

```
SELECT COUNT(*) AS 'Total Married' FROM developers WHERE married is true;
```

To get the total number of developers who can code in Python:

```
SELECT COUNT(*) AS 'Python Pros' FROM developers WHERE INSTR(languages,"python") > 0;
```

To find the total number of unique developer names:

```
SELECT COUNT(DISTINCT name) AS 'Total names' FROM developers;
```

8 Adding, updating and deleting rows in a table

We can add a row to a table by specifying the literal values for all the columns as they are specified in the table's structure (as given by `describe developers;`)

```
INSERT INTO developers
VALUES (null, 48, 'php,python', true, 'ironman');
```

Since the first column id is the primary key which is set to AUTO_INCREMENT, passing it the value of null will cause this new row to placed right at the end of the table with its id value incremented by 1 from the value in the last row.

Verify this with:

```
SELECT * FROM developers;
```

We can also add a new row by specifying values for some of the columns (rather than all at once):

```
INSERT INTO developers(name, age)
VALUES ('thanos', 2000);
```

In this case, the new row is also created at the bottom of the existing table and its unspecified columns will have the value of NULL. This can be later set using the update operation.

We can update the column values of any particular row in a table by selecting it based on its id:

Let's first retrieve the row with the id of 5 to check its contents:

```
SELECT * FROM developers WHERE id=5;
```

We can update one or more columns in this row with:

```
UPDATE developers SET name = 'Spiderman', age = 33
WHERE id = 5;
```

Confirm the update by retrieving the same records again:

```
SELECT * FROM developers WHERE id=5;
```

We can also update column values of multiple rows in table that are filtered with the WHERE criteria.
For e.g. we can update the name of all developers who can code in Java with:

```
UPDATE developers SET name = 'Super Developer'
WHERE INSTR(languages,"java") > 0;
```

Verify the update with:

```
SELECT name, languages FROM developers WHERE INSTR(languages,"java")
> 0;
```

Similarly, we can delete a single record with:

```
DELETE FROM developers WHERE id = 10;
```

And then verify that the row is gone by attempting to retrieve it and obtaining an empty result:

```
SELECT * FROM developers WHERE id = 10;
```

We can delete multiple rows in a table that are filtered with the WHERE criteria.
For e.g. we can delete all developers who can code in Java with:

```
DELETE FROM developers WHERE INSTR(languages,"java") > 0;
```

and then verify that all these rows are gone by attempting to retrieve them and obtaining an empty result instead:

```
SELECT * FROM developers WHERE INSTR(languages,"java") > 0;
```

9 Creating and deleting a table

To delete an entire table, simply execute:

```
DROP TABLE developers;
```

To create a new table, specify the name of the table and the data types of the various columns.
For e.g.

```
CREATE TABLE IF NOT EXISTS employees (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
```

```
    start_date DATE,
    salary FLOAT
);
```

The complete list of MySQL datatypes

To verify the table creation, check its structure with:

```
DESCRIBE employees;
```

We can add some rows into this newly created table with:

```
INSERT INTO employees VALUES
(null, 'Superman', '1990-12-27', 1200.12),
(null, 'Ironman', '2010-06-01', 2300.88),
(null, 'Wonder Woman', '2018-10-15', 3500.67);
```

Verify that the row have been successfully added with:

```
SELECT * FROM employees;
```

10 Exporting a table from a database

Check the currently active database holding the table to be exported:

```
SELECT DATABASE() FROM DUAL;
```

In a separate normal command prompt, navigate to an empty directory and execute:

```
mysqldump -u root -p workshopdb employees > emptable.sql
```

You will again be prompted for the admin (root) password. Verify that the `emptable.sql` is created here. If you check its contents, it consists primarily of SQL statements to create the table, populate it and some other relevant preparatory actions.

Back in the MySQL command line client, you can delete this table:

```
DROP TABLE employees;
```

And reimport it again from the command prompt using the command we executed earlier at the start of this lab:

```
mysql -u root -p workshopdb < emptable.sql
```

You can find more comprehensive tutorials on MySQL at:

https://www.mysqltutorial.org/
https://dev.mysql.com/doc/refman/8.4/en/tutorial.html