Querying Data

This lesson teaches how to query for data in MySQL.

Querying Data

In this lesson we'll learn how to query the data we have stored in a table. The **SELECT** statement allows us to retrieve data from tables.

Example Syntax

> SELECT **col1, col2, ... coln**
>
> FROM **table**
>
> WHERE <**condition**>

> Connect to the terminal below by clicking in the widget. Once connected, the command line prompt will show up. Enter or copy and paste the command **./DataJek/Lessons/8lesson.sh** and wait for the MySQL prompt to start-up.

```
-- The lesson queries are reproduced below for convenient copy/paste into the terminal.

-- Query 1
SELECT * from Actors;

-- Query 2
SELECT <columns> FROM <TableName>

-- Query 3
SELECT FirstName, SecondName from Actors;

-- Query 4
```

```
SELECT FirstName, SecondName from Actors WHERE FirstName="Travolta";

-- Query 5

SELECT FirstName, SecondName from Actors WHERE FirstName="Brad";

-- Query 6
SELECT FirstName, SecondName from Actors WHERE NetWorthInMillions > 500;

-- Query 7
SELECT FirstName, SecondName from Actors WHERE NetWorthInMillions > 0;
```

Terminal

1. Execute the following **SELECT** statement to retrieve all the rows in the table with all the columns.

```
SELECT * from Actors;
```

```
mysql> SELECT * FROM Actors;
+----+-----------+------------+------------+--------+---------------+--------------------+
| Id | FirstName | SecondName | DoB        | Gender | MaritalStatus | NetWorthInMillions |
+----+-----------+------------+------------+--------+---------------+--------------------+
|  1 | Brad      | Pitt       | 1963-12-18 | Male   | Single        |                240 |
|  2 | Jennifer  | Aniston    | 1969-11-02 | Female | Single        |                240 |
|  3 | Angelina  | Jolie      | 1975-06-04 | Female | Single        |                100 |
|  4 | Johnny    | Depp       | 1963-06-09 | Male   | Single        |                200 |
|  5 | Natalie   | Portman    | 1981-06-09 | Male   | Married       |                 60 |
|  6 | Tom       | Cruise     | 1962-07-03 | Male   | Divorced      |                570 |
|  7 | Kylie     | Jenner     | 1997-08-10 | Female | Married       |               1000 |
|  8 | Kim       | Kardashian | 1980-10-21 | Female | Married       |                370 |
|  9 | Amitabh   | Bachchan   | 1942-10-11 | Male   | Married       |                400 |
| 10 | Shahrukh  | Khan       | 1965-11-02 | Male   | Married       |                600 |
| 11 | priyanka  | Chopra     | 1982-07-18 | Female | Married       |                 28 |
+----+-----------+------------+------------+--------+---------------+--------------------+
11 rows in set (0.00 sec)
```

The command reads all the data in the table. The syntax for a simple **SELECT** command is as follows:

```
SELECT <columns> FROM <TableName>
```

The **SELECT** keyword is followed by a comma-separated list of columns we wish to display. Using an * displays all the columns. The table to query is specified using the **FROM** keyword followed by the table name.

2. In the next command, we'll display the first name and second name columns. Execute the following command:

```
SELECT FirstName, SecondName from Actors;
```

The outcome is as follows:

```
mysql> SELECT FirstName, SecondName from Actors;
+-----------+------------+
| FirstName | SecondName |
+-----------+------------+
| Brad      | Pitt       |
| Jennifer  | Aniston    |
| Angelina  | Jolie      |
| Johnny    | Depp       |
| Natalie   | Portman    |
| Tom       | Cruise     |
| Kylie     | Jenner     |
| Kim       | Kardashian |
| Amitabh   | Bachchan   |
| Shahrukh  | Khan       |
| priyanka  | Chopra     |
+-----------+------------+
11 rows in set (0.00 sec)
```

The columns are displayed in the same order as they appear in the MYSQL query.

3. We can filter the rows for a select query using the **WHERE** clause. The **WHERE** clause specifies a criterion that rows must match to be returned by the **SELECT** query. The criteria may be met by zero, one, multiple, or all rows.

   Executing the following query will result in no row being matched:

   ```
   SELECT FirstName, SecondName from Actors WHERE FirstName="Travolt
   a";
   ```

   ```
   mysql> SELECT FirstName, SecondName from Actors WHERE FirstName="Travolta";
   Empty set (0.00 sec)
   ```

   Executing the following query will result in exactly one row being matched:

   ```
   SELECT FirstName, SecondName from Actors WHERE FirstName="Brad";
   ```

```
mysql> SELECT FirstName, SecondName from Actors WHERE FirstName="Brad";
+-----------+------------+
| FirstName | SecondName |
+-----------+------------+
| Brad      | Pitt       |
+-----------+------------+
1 row in set (0.00 sec)
```

Executing the following query results in multiple rows being matched:

```
SELECT FirstName, SecondName from Actors WHERE NetWorthInMillion
s > 500;
```

```
mysql> SELECT FirstName, SecondName from Actors WHERE NetWorthInMillions > 500;
+-----------+------------+
| FirstName | SecondName |
+-----------+------------+
| Tom       | Cruise     |
| Kylie     | Jenner     |
| Shahrukh  | Khan       |
+-----------+------------+
3 rows in set (0.00 sec)
```

Finally, executing the following query results in all rows being matched and returned:

```
SELECT FirstName, SecondName from Actors WHERE NetWorthInMillion
s > 0;
```

```
mysql> SELECT FirstName, SecondName from Actors WHERE NetWorthInMillions > 0;
+-----------+------------+
| FirstName | SecondName |
+-----------+------------+
| Brad      | Pitt       |
| Jennifer  | Aniston    |
| Angelina  | Jolie      |
| Johnny    | Depp       |
| Natalie   | Portman    |
| Tom       | Cruise     |
| Kylie     | Jenner     |
| Kim       | Kardashian |
| Amitabh   | Bachchan   |
| Shahrukh  | Khan       |
| priyanka  | Chopra     |
+-----------+------------+
11 rows in set (0.00 sec)
```

The following table captures the various operators that can be used in a **WHERE** clause.

Operator	Purpose
>	Greater than operator
>=	Greater than or equal to operator
<	Less than operator
<=	Less than or equal to operator
!=	Not equal operator
<>	Not equal operator
<=>	NULL-safe equal to operator
=	Equal to operator
BETWEEN ... AND ...	Whether a value is within a range of values
COALESCE()	Return the first non-NULL argument
GREATEST()	Return the largest argument
IN	Whether a value is within a set of values
INTERVAL	Return the index of the argument that is less than the first argument
IS	Test a value against a boolean

IS NOT	Test a value against a boolean
IS NOT NULL	NOT NULL value test
IS NULL	NULL value test
ISNULL()	Test whether the argument is NULL
LEAST()	Return the smallest argument
LIKE	Simple pattern matching
NOT BETWEEN … AND …	Whether a value is not within a range of values
NOT IN()	Whether a value is not within a set of values
NOT LIKE	Negation of simple pattern matching
STRCMP()	Compare two strings