COURSEWARE

Data Query Language using SELECT

Contents

Overview

Data Query Language (DQL) allows us to read data, both about the schema (DDL) and data (DML) from a database.

DQL contains a single command - `SELECT`, which allows us to retrieve information from a database.

Arguably, `SELECT` is the most powerful command within SQL, due to its versatility.

Technically, DQL is a subsection of DML; therefore, it is fair to say that the `SELECT` statement is either DML or DQL.

Recap: Reading with DDL

DDL allows us to read and retrieve information about our schema.

As the schema is the focus and we aren't manipulating the data within our database, this falls into DDL.

To see what databases are in our entire MySQL server, use the command:

```
SHOW databases;
```

We can also see all tables within a database we are using with the below command;

```
SHOW tables;
```

Finally, we can see all of the properties of a specified table with the below;

```
DESCRIBE table_name;
```

Reading with SELECT

The `SELECT` statement is used to read data from a database.

We can use this in a very simplistic way by selecting all records from a table, or add a number of different clauses, grouping and sorting options.

The syntax for a `SELECT` statement is similar to the below;

```
SELECT field_name1, field_name2 FROM table_name;
```

If you want to see all of the records from a table, we can run the below;

```
SELECT * FROM table_name;
```

SELECT DISTINCT

We can add a number of other keywords to our `SELECT` statement to make the data returned more specific to what we need

One way of doing this is by using the `DISTINCT` keyword, which will only return unique values of a particular column.

For instance, perhaps we want to see all of the cities which customers live in.

We could run something similar to the below and manually find all unique entries;

```
SELECT city FROM customers
```

For a case like this, we can use the `DISTINCT` keyword which, when attached to a field, will find all of the unique values in that column.

```
SELECT DISTINCT city FROM customers
```

The WHERE clause

If we wish to add further specifics and criteria to what we want to read from a table, we can use the `WHERE` clause in our `SELECT` statement.

```
SELECT fields FROM table_name WHERE condition;
```

For example, if we wanted to see all of the products that are over £50;

```
SELECT * FROM products WHERE price > 50;
```

WHERE clause operators

The `WHERE` clause can also use several different operators for comparisons:

- **=** (equal), **!=** (not equal)

```
SELECT * FROM products WHERE product_name = 'Stevens Watch';
```

```
SELECT * FROM customers WHERE city != 'Manchester';
```

- `<` and `>` (less and greater than), `<=` and `>=` (less than or equal, greater than or equal)

```
SELECT * FROM products WHERE price < 10;
```

```
SELECT * FROM customers WHERE age >= 18;
```

- `BETWEEN` – within an inclusive range

```
SELECT * FROM orders WHERE datePlaced BETWEEN '2019-01-01' AND '2019-12-31';
```

- `LIKE` – searching for a pattern

```
SELECT * FROM customers WHERE name LIKE '%son';
```

- `(NOT) IN` – specifying multiple possible values for a column

```
SELECT * FROM customers WHERE city IN ('Manchester', 'London', 'Birmingham');
```

- `IS (NOT) NULL` – select everything where the specified field is(n't) null

```
SELECT * FROM customers WHERE email IS NULL;
```

Aliasing

We can use the aliasing keyword `AS` to present our data differently. It is commonly used for renaming columns on which some kind of arithmetic has been done:

```
SELECT field_name1 AS name, field_name2 FROM table_name;
```

For example, if we wanted to calculate a price within a query, we could use the below as an example;

```
SELECT title, quantity, price, quantity*price AS stock_value FROM products;
```

Ordering Data

The `ORDER BY` keyword allows us to sort our records upon retrieval.

`ORDER BY` will, by default, present selected data in ascending order unless specified within the query;

```
SELECT field_name1, field_name2 FROM table_name ORDER BY field_name2 DESC;
```

If you wish to specify for MySQL to order ascending, you can do so with the below;

```
SELECT field_name1, field_name2 FROM table_name ORDER BY field_name2 ASC;
```

To do so for descending order, use the `DESC` keyword instead.

Limiting Data

We can use the `LIMIT` keyword alongside `ORDER BY` to see a specific snapshot of the data we want.

By using `LIMIT`, we can see just a few records outputted rather than an entire set.

```
SELECT * FROM customers LIMIT 1;
```

Very useful for answering questions like “what are the top 5 x in table y?”

```
SELECT * FROM products ORDER BY price DESC LIMIT 5;
```

Tutorial

There is no tutorial for this module.

Exercises

Test out some `SELECT` statements on some of the columns from your tables.

(*note: if you have not created records for your database yet, refer back to the Data Manipulation Language module for context*)

▶ Solution