SQL DML. Part 1.

Single table queries

- This topic is split into 5 sections
  - Single Table Queries
  - Aggregate Functions
  - Joins
  - Sub-queries
  - Groups
- For each section there will be
  - Slides
  - An output from the example queries on the slides
  - A tutorial with written explanations and questions for you to try
  - Tutorial answers
  - Homework

Objectives of SQL

CITY UNIVERSITY LONDON
EST 1894

- Ideally, database language should allow user to:
  - create the database and relation structures;
  - perform insertion, modification, deletion of data from relations;
  - perform simple and complex queries.
- Must perform these tasks with minimal user effort and command structure/syntax must be easy to learn.
- It must be portable.

Objectives of SQL

CITY UNIVERSITY LONDON

SQL is a transform-oriented language with two major components:

- A DDL (Data Definition Language) for defining database structure.
- A DML (Data Manipulation Language) for retrieving and updating data.

Objectives of SQL

CITY UNIVERSITY LONDON

SQL is relatively easy to learn:

- it is non-procedural - you specify what information you require, rather than how to get it;
- it is essentially free-format.

Writing SQL Queries

CITY UNIVERSITY LONDON

SQL statement consists of reserved words and user-defined words.

- Reserved words are a fixed part of SQL and must be spelt exactly as required and cannot be split across lines.
- User-defined words are made up by user and represent names of various database objects such as relations, columns, views.

SELECT Statement

CITY UNIVERSITY LONDON

```
SELECT [DISTINCT | ALL]
    {* | [columnExpression [AS newName]] [,...] }
FROM   TableName [alias] [, ...]
[WHERE condition]
[GROUP BY  columnList]  [HAVING   condition]
[ORDER BY  columnList]
```

Used database

CITY UNIVERSITY LONDON
EST 1894

For this lecture and tutorial sessions we will use the mcc database.

The setup file can be found on Moodle: mcc_setup.sql.

`mccPlayer(registration_number` **{pk}**`, name, age, p_street, p_town, p_postcode)`

`mccMatches(Match_date` **{pk}**`, opposing_team, ground_name)`

`mccGround(ground_name` **{pk}**`, g_street, g_town, g_postcode)`

`mccMatch_performance(registration_number` **{pk}**`, match_date` **{pk}**`, batting_score)`

Primary keys are shown **{pk}**.

Note. In this database, the dates are represented as numbers in the form `yymmdd`, e.g. the `26th May 2013` is `130526`. `Batting_score` and `age`, are also numeric but all other attributes are character strings.

All Columns, All Rows

CITY UNIVERSITY LONDON

List full details of all players.

```
SELECT  registration_number, name, age, p_street, p_town, p_postcode
    FROM mccPlayer;
```

or

Can use * as an abbreviation for ‘all columns’:

```
SELECT * FROM mccPlayer;
```

Specific Columns, All Rows

CITY UNIVERSITY LONDON

Produce a list of names and towns of players.

```
SELECT name, p_town FROM mccPlayer;
```

Name as “Full_name” and “Town”.

```
SELECT name AS "Full_name", p_town AS "Town" FROM mccPlayer;
```

Use of DISTINCT

CITY UNIVERSITY LONDON

Compare these two queries:

```
SELECT p_town FROM mccPlayer;
```

```
SELECT DISTINCT p_town FROM mccPlayer;
```

```
SELECT DISTINCT p_town AS Town FROM mccPlayer;
```

Conditions. Numeric Comparison.

CITY UNIVERSITY LONDON

List the name and ages of all players older than 35

```
SELECT name, age
FROM mccPlayer
WHERE age > 35;
```

Conditions. Numeric Comparison.

CITY UNIVERSITY LONDON

List the names and ages of all players under 30 years old

```
SELECT name, age
FROM mccPlayer
WHERE age < 30;
```

Conditions. Numeric Comparison.

CITY UNIVERSITY LONDON

List the names and ages of all players not younger than 35 or under 30 years old

```
SELECT name, age
FROM mccPlayer
WHERE age >= 35 OR age < 30;
```

Conditions. String Comparison.

CITY UNIVERSITY LONDON

List the names and ages of all players aged 35 or over who live in Alnwick

```
SELECT name, age
FROM mccPlayer
WHERE age > 35
AND p_town = 'Alnwick';
```

And those who don't live in Alnwick

```
SELECT name, age
FROM mccPlayer
WHERE age > 35
AND p_town != 'Alnwick';
```

Conditions. Set Membership.

CITY UNIVERSITY LONDON

List the names, ages, and town of all players that live in Alnwick or Morpeth

```
SELECT name, age, p_town
FROM mccPlayer
WHERE p_town IN ("Alnwick", "Morpeth");
```

And who do not live in Alnwick of Morpeth

```
SELECT name, age, p_town
FROM mccPlayer
WHERE p_town NOT IN ("Alnwick", "Morpeth");
```

Conditions. Pattern Matching.

CITY UNIVERSITY LONDON

- SQL has two special pattern matching symbols:
  - %: sequence of zero or more characters;
  - _ (underscore): any single character.
- `LIKE "%York%"` means a sequence of characters of any length containing "York". For example, `"Yorkshire"` and `"New York"` satisfy.

Conditions. Pattern Matching.

CITY UNIVERSITY LONDON
EST 1894

```
SELECT name, age, p_town
FROM mccPlayer
WHERE p_town LIKE "Alnwick";

SELECT name, age, p_town
FROM mccPlayer
WHERE p_town LIKE "%wick";

SELECT name, age, p_town
FROM mccPlayer
WHERE p_town LIKE "%r%";

SELECT name, age, p_town
FROM mccPlayer
WHERE p_town LIKE "__r%";
```

List the name of players that are from towns with “r” and “u”, and then with “r” after “e”

Conditions. Pattern Matching.

CITY UNIVERSITY LONDON

```sql
SELECT name, age, p_town
FROM mccPlayer
WHERE p_town LIKE "%r%" AND p_town LIKE "%u%";
```

```sql
SELECT name, age, p_town
FROM mccPlayer
WHERE p_town LIKE "%e%r%";
```

Conditions. NULL Values.

CITY UNIVERSITY LONDON

```
INSERT INTO mccPlayer VALUES
    ("1013", "Chris Smart", 50, NULL, "Tynemouth", "NE30 1RP");


SELECT *
FROM mccPlayer
WHERE p_street IS NULL;

SELECT *
FROM mccPlayer
WHERE p_street IS NOT NULL;
```

Ordering Results

CITY UNIVERSITY LONDON

```sql
SELECT name, age, p_town
FROM mccPlayer
ORDER BY p_town ASC;
```

```sql
SELECT name, age, p_town
FROM mccPlayer
ORDER BY p_town DESC;
```

```sql
SELECT ground_name, g_town
FROM mccGround
ORDER BY g_town DESC, ground_name ASC;
```