Databases and Info Systems

Structured Query Language (SQL)

Dr. Seán Russell
sean.russell@ucd.ie,

School of Computer Science,
University College Dublin

February 26, 2020

Table of Contents

Example Tables

employees

emp_id	name	title	salary	dept_id	join_date
1234	Sean Russell	Trainer	50000	10	2018-03-01
4567	Jamie Heaslip	Manager	47000	10	2004-10-21
6542	Leo Cullen	Trainer	45000	10	2012-12-01
1238	Brendan Macken	Technician	25000	20	2001-09-10
1555	Sean O'Brien	Designer	50000	20	1999-06-24
1899	Brian O'Driscoll	Manager	45000	20	1998-02-27
2525	Peter Stringer	Designer	25000	30	2017-01-16
1585	Denis Hickey	Architect	20000	30	2009-08-07
1345	Ronan O'Gara	Manager	29000	30	2019-12-25

departments

dept_id	dept_name	office	division	manager_id
10	Training	Lansdowne	D1	4567
20	Design	Belfield	D2	1899
30	Implementation	Donnybrook	D1	1345
40	Strategy	Terenure	D2	NULL

Example Tables

appointments

title	date	start_time	end_time
Head of School	2020-02-25	15:20:00	15:40:00
Football	2020-02-25	17:30:00	19:00:00
Sleeping	2020-02-25	22:30:00	

- These tables are available in the file `week4.db` on moodle

Importing a Database

- To import a database from a file, we must complete the following steps
  1. Log in to mysql using your username and password (make sure the file is in the same folder as your command prompt)
  2. Create the database e.g. CREATE DATABASE week4;
  3. Select the database e.g. USE week4
  4. Load the file e.g. source week4.db
- Now all of the tables and the data stored in them will be loaded and ready to query

SQL Select Query Syntax

- SQL queries are expressed by the SELECT statement.
- The basic syntax is:

```
SELECT attr_expr { , attr_expr } FROM table_name { ,
    table_name } [ WHERE condition ] ;
```

  - [] means an optional expression
  - {} means an optional list of expressions

Select Query Parts

```
SELECT attr_expr {, attr_expr } FROM table_name {,
    table_name } [ WHERE condition ] ;
```

- The three parts of the query are usually called:
  - Target list (the attributes you want to retrieve, and/or expressions based on these attributes).
  - From clause (the table(s) to select from)
  - Where clause (the condition on which to select rows)

Order

- Queries are completed in a particular order
  1. The Cartesian product of the tables in the **FROM** clause is generated
  2. The result is restricted to the rows that satisfy the condition in there **WHERE** clause
  3. The attribute expressions in the target list are evaluated

Simple Query Examples

```
1 SELECT salary FROM employees WHERE title =
    "Technician";
```

```
1 +--------+
2 | salary |
3 +--------+
4 |  25000 |
5 +--------+
6 1 row in set (0.00 sec)
```

Simple Query Examples

```
1 SELECT * FROM employees WHERE title =
    "Manager";
```

```
1 +--------+--------------------+---------+--------+---------+------------+
2 | emp_id | name               | title   | salary | dept_id | join_date  |
3 +--------+--------------------+---------+--------+---------+------------+
4 | 1345   | Ronan O'Gara       | Manager |  29000 | 30      | 2019-12-25 |
5 | 1899   | Brian O'Driscoll   | Manager |  45000 | 20      | 1998-02-27 |
6 | 4567   | Jamie Heaslip      | Manager |  47000 | 10      | 2004-10-21 |
7 +--------+--------------------+---------+--------+---------+------------+
8 3 rows in set (0.00 sec)
```

Table of Contents

Comparison Operators

- SQL allows the use of the typical comparison operators
  - $=$ Equality
  - $<$ Less than
  - $>$ Greater than
  - $<=$ Less than or equal
  - $>=$ Greater than or equal
  - $<>$ Not equal
  - $!=$ Not equal

Complex WHERE Clauses

- The **WHERE** clause is a boolean expression
  - it is evaluated for every row in the table
- We can use boolean operators **AND**, **OR**, **NOT** to combine simple expressions into more complex ones
- Find all managers who earn at least 30000

```
1 SELECT * FROM employees WHERE
    title="Manager" AND salary>=30000;
```

```
1 +--------+-------------------+---------+--------+---------+------------+
2 | emp_id | name              | title   | salary | dept_id | join_date  |
3 +--------+-------------------+---------+--------+---------+------------+
4 | 1899   | Brian O'Driscoll  | Manager |  45000 | 20      | 1998-02-27 |
5 | 4567   | Jamie Heaslip     | Manager |  47000 | 10      | 2004-10-21 |
6 +--------+-------------------+---------+--------+---------+------------+
7 2 rows in set (0.00 sec)
```

Brackets in Where Clause

- You can use parentheses (round brackets) to group boolean statements correctly
- Find all managers who work in either department 10 or department 20

```
SELECT * FROM employees WHERE
   title="Manager" AND (dept_id="10" OR
   dept_id="20");
```

```
+--------+--------------------+---------+--------+---------+------------+
| emp_id | name               | title   | salary | dept_id | join_date  |
+--------+--------------------+---------+--------+---------+------------+
| 1899   | Brian O'Driscoll   | Manager |  45000 | 20      | 1998-02-27 |
| 4567   | Jamie Heaslip      | Manager |  47000 | 10      | 2004-10-21 |
+--------+--------------------+---------+--------+---------+------------+
2 rows in set (0.00 sec)
```

Comparison Functions

- Full list of comparison functions and operators in MySQL:
  - https://dev.mysql.com/doc/refman/8.0/en/comparison-operators.html
- Interesting operators:
  - BETWEEN ... AND
  - IN

BETWEEN ... AND

- Search for values in a particular range
- Find all employees paid between 25000 and 40000

```
SELECT * FROM employees WHERE salary BETWEEN 25000
    AND 40000;
```

- Is the same as

```
SELECT * FROM employees WHERE salary >= 25000 AND
    salary <= 40000;
```

```
+--------+-----------------+------------+--------+---------+------------+
| emp_id | name            | title      | salary | dept_id | join_date  |
+--------+-----------------+------------+--------+---------+------------+
| 1238   | Brendan Macken  | Technician |  25000 | 20      | 2001-09-10 |
| 1345   | Ronan O'Gara    | Manager    |  29000 | 30      | 2019-12-25 |
| 2525   | Peter Stringer  | Designer   |  25000 | 30      | 2017-01-16 |
+--------+-----------------+------------+--------+---------+------------+
3 rows in set (0.00 sec)
```

IN

- Check if a value is contained in a set of values
- Find all employees with the title `Trainer`, `Designer`, or `Architect`

```
SELECT * FROM employees WHERE title IN
    ("Trainer","Designer","Architect");
```

- Same as the clause `WHERE title="Trainer" OR title="Designer" OR title="Architect"`

```
+--------+---------------+-----------+--------+---------+------------+
| emp_id | name          | title     | salary | dept_id | join_date  |
+--------+---------------+-----------+--------+---------+------------+
| 1234   | Sean Russell  | Trainer   |  50000 | 10      | 2018-03-01 |
| 1555   | Sean O'Brien  | Designer  |  50000 | 20      | 1999-06-24 |
| 1585   | Denis Hickey  | Architect |  20000 | 30      | 2009-08-07 |
| 2525   | Peter Stringer| Designer  |  25000 | 30      | 2017-01-16 |
| 6542   | Leo Cullen    | Trainer   |  45000 | 10      | 2012-12-01 |
+--------+---------------+-----------+--------+---------+------------+
5 rows in set (0.00 sec)
```

String Functions

- There are many functions that we can apply to attributes in the WHERE clause
- A full list of the MYSQL String functions can be found here:
  - https://dev.mysql.com/doc/refman/8.0/en/string-functions.html
- Interesting functions
  - LIKE
  - LENGTH
  - RIGHT

LIKE

- Search for patterns in string data
- Find all employees where the second letter of their name is e

```
1 SELECT * FROM employees WHERE name LIKE "_e%";
```

  - "_" represents a single character
  - "%" represents any number of characters

```
1  +--------+-----------------+-----------+--------+---------+------------+
2  | emp_id | name            | title     | salary | dept_id | join_date  |
3  +--------+-----------------+-----------+--------+---------+------------+
4  | 1234   | Sean Russell    | Trainer   |  50000 | 10      | 2018-03-01 |
5  | 1555   | Sean O'Brien    | Designer  |  50000 | 20      | 1999-06-24 |
6  | 1585   | Denis Hickey    | Architect |  20000 | 30      | 2009-08-07 |
7  | 2525   | Peter Stringer  | Designer  |  25000 | 30      | 2017-01-16 |
8  | 6542   | Leo Cullen      | Trainer   |  45000 | 10      | 2012-12-01 |
9  +--------+-----------------+-----------+--------+---------+------------+
10 5 rows in set (0.00 sec)
```

LIKE

- Find all employees who's job title contains exactly seven letters

```
SELECT * FROM employees WHERE title LIKE "_______";
```

```
+--------+-----------------+---------+--------+---------+------------+
| emp_id | name            | title   | salary | dept_id | join_date  |
+--------+-----------------+---------+--------+---------+------------+
| 1234   | Sean Russell    | Trainer |  50000 | 10      | 2018-03-01 |
| 1345   | Ronan O'Gara    | Manager |  29000 | 30      | 2019-12-25 |
| 1899   | Brian O'Driscoll| Manager |  45000 | 20      | 1998-02-27 |
| 4567   | Jamie Heaslip   | Manager |  47000 | 10      | 2004-10-21 |
| 6542   | Leo Cullen      | Trainer |  45000 | 10      | 2012-12-01 |
+--------+-----------------+---------+--------+---------+------------+
5 rows in set (0.00 sec)
```

LENGTH

- Get the length of a string in an attribute
- Find all employees who's job title contains exactly seven letters

```
SELECT * FROM employees WHERE LENGTH(title)=7;
```

```
+--------+-----------------+---------+--------+---------+------------+
| emp_id | name            | title   | salary | dept_id | join_date  |
+--------+-----------------+---------+--------+---------+------------+
| 1234   | Sean Russell    | Trainer |  50000 | 10      | 2018-03-01 |
| 1345   | Ronan O'Gara    | Manager |  29000 | 30      | 2019-12-25 |
| 1899   | Brian O'Driscoll| Manager |  45000 | 20      | 1998-02-27 |
| 4567   | Jamie Heaslip   | Manager |  47000 | 10      | 2004-10-21 |
| 6542   | Leo Cullen      | Trainer |  45000 | 10      | 2012-12-01 |
+--------+-----------------+---------+--------+---------+------------+
5 rows in set (0.00 sec)
```

RIGHT (and LEFT)

- The `RIGHT(str,num)` function returns the last `num` characters of the string `str`
- Find all employees whose names end in "ll"

```
SELECT * FROM employees WHERE RIGHT(name,2)="ll";
```

```
+--------+-----------------+---------+--------+---------+------------+
| emp_id | name            | title   | salary | dept_id | join_date  |
+--------+-----------------+---------+--------+---------+------------+
| 1234   | Sean Russell    | Trainer |  50000 | 10      | 2018-03-01 |
| 1899   | Brian O'Driscoll| Manager |  45000 | 20      | 1998-02-27 |
+--------+-----------------+---------+--------+---------+------------+
2 rows in set (0.00 sec)
```

Date and Time Functions

- Other data types also have available functions
- A full list of the MySQL date and time functions can be found here:
  - https://dev.mysql.com/doc/refman/8.0/en/date-and-time-functions.html
- Interesting functions
  - `CURDATE()`
  - `CURTIME()`
  - `NOW()`
  - `DATEDIFF`

CURDATE

- Gets the current date (also works in target list)

```
SELECT CURDATE();
```

```
+------------+
| CURDATE()  |
+------------+
| 2020-02-25 |
+------------+
1 row in set (0.00 sec)
```

- Find employees who were hired today

```
SELECT * FROM employees WHERE join_date =
    CURDATE();
```

CURTIME

- Gets the current time (also works in target list)

```
SELECT CURTIME();
```

```
+-------------+
| CURTIME()   |
+-------------+
| 17:01:07    |
+-------------+
1 row in set (0.00 sec)
```

- Find all appointments today that have finished:

```
SELECT * FROM appointments WHERE date = CURDATE()
    AND end_time < CURTIME();
```

```
+-----------------+------------+------------+----------+
| title           | date       | start_time | end_time |
+-----------------+------------+------------+----------+
| Head of School  | 2020-02-25 | 15:20:00   | 15:40:00 |
| Football        | 2020-02-25 | 17:30:00   | 19:00:00 |
+-----------------+------------+------------+----------+
2 rows in set (0.00 sec)
```

NOW

- Work very similarly to CURTIME, but returns a DATETIME object instead of a TIME object
- Works in both target list and WHERE clause

DATEDIFF

- This function returns the number of days between two DATE or DATETIME values
- Syntax: DATEDIFF(expr1,expr2)
- Find all employees who have joined in the last 90 days

```
SELECT * FROM employees WHERE DATEDIFF(CURDATE(),
    join_date) < 90;
```

```
+--------+----------------+---------+--------+---------+------------+
| emp_id | name           | title   | salary | dept_id | join_date  |
+--------+----------------+---------+--------+---------+------------+
| 1345   | Ronan O'Gara | Manager |  29000 | 30      | 2019-12-25 |
+--------+----------------+---------+--------+---------+------------+
1 row in set (0.00 sec)
```

Expressions in Target List

- Expressions can be used in the target list
- Assuming the database contains annual salary, find the monthly salary of every employee

```
SELECT name, salary/12 FROM employees;
```

```
+--------------------+-------------+
| name               | salary/12   |
+--------------------+-------------+
| Sean Russell       | 4166.6667   |
| Brendan Macken     | 2083.3333   |
| Ronan O'Gara       | 2416.6667   |
| Sean O'Brien       | 4166.6667   |
| Denis Hickey       | 1666.6667   |
| Brian O'Driscoll   | 3750.0000   |
| Peter Stringer     | 2083.3333   |
| Jamie Heaslip      | 3916.6667   |
| Leo Cullen         | 3750.0000   |
+--------------------+-------------+
9 rows in set (0.00 sec)
```

Expressions in Target List

- The previous example gave ugly output
- We can use a function to round the answer
  - There are other functions available here:
  - https://dev.mysql.com/doc/refman/8.0/en/numeric-functions.html

```
1 SELECT name, ROUND(salary/12) FROM employees;
```

```
1 +-------------------+--------------------+
2 | name              | ROUND(salary/12)   |
3 +-------------------+--------------------+
4 | Sean Russell      |               4167 |
5 ...
6 +-------------------+--------------------+
7 9 rows in set (0.00 sec)
```

Table of Contents

Aliases

- Using expressions and functions in the target list gives us complex headings for our results
- We can use an alias to temporarily give a descriptive name

```
1 SELECT name, ROUND(salary/12) AS month_salary FROM
      employees;
```

```
1 +--------------------+--------------+
2 | name               | month_salary |
3 +--------------------+--------------+
4 | Sean Russell       |         4167 |
5 ...
6 +--------------------+--------------+
7 9 rows in set (0.00 sec)
```

Aliases on Tables

- We can also use aliases for tables to shorten our queries
- This becomes very useful for complex joins

```
1 SELECT name, dept_name FROM employees, departments
    WHERE employees.dept_id=departments.dept_id;
```

- Is the same as:

```
1 SELECT name, dept_name FROM employees AS e,
    departments AS d WHERE e.dept_id=d.dept_id;
```

- This is more useful the more joins you have

More Complex Joins

- One of the big challenges of using databases is translating a requirement in human language and writing a suitable SQL query to answer it

- Find the names of the employees who work in the Lansdowne office of division D1

More Complex Joins

- Find the names of the employees who work in the Lansdowne office of division D1
- Query:

```
SELECT name FROM employees AS e, departments AS d
    WHERE e.dept_id = d.dept_id AND d.division =
    'D1' AND d.office = 'Lansdowne';
```

- Result:

```
+-----------------+
| name            |
+-----------------+
| Sean Russell    |
| Jamie Heaslip   |
| Leo Cullen      |
+-----------------+
3 rows in set (0.00 sec)
```

More Complex Joins

- Find the names of the employees who work in either the Lansdowne office or the Belfield office
- Query 1:

```
1 SELECT name FROM employees AS e, departments AS d
    WHERE e.dept_id = d.dept_id AND
    (d.office="Belfield" OR d.office="Lansdowne");
```

- Query 2:

```
1 SELECT name FROM employees AS e, departments AS d
    WHERE e.dept_id = d.dept_id AND d.office
    IN("Belfield","Lansdowne");
```

Table of Contents

NULL Values

- NULL values may mean that:
  - a value is not applicable
  - a value is applicable but unknown
  - it is unknown if a value is applicable or not

Comparing NULL Values

- Previous standards of SQL used two-valued logic
  - Only the values TRUE and FALSE
  - Comparing against a NULL value returns FALSE
- SQL-2 (and later) use a three-valued logic
  - Uses the values TRUE, FALSE and UNKNOWN
  - Comparing against a NULL value returns UNKNOWN

Testing for NULL Values

- In a SELECT query, we may want to test if an attribute contains a NULL value.
  - attribute IS NULL
  - attribute IS NOT NULL
- Find the names of all departments that do not have a manager

```
1 SELECT dept_name FROM departments WHERE manager_id
      IS NULL;
```

```
1 +-------------+
2 | dept_name   |
3 +-------------+
4 | Strategy    |
5 +-------------+
6 1 row in set (0.00 sec)
```

Testing for NOT NULL Values

- In a SELECT query, we may want to test if an attribute contains a NULL value.
  - attribute IS NULL
  - attribute IS NOT NULL
- Find the names of all departments that have a manager

```
1 SELECT dept_name FROM departments WHERE manager_id
      IS NOT NULL;
```

```
1 +-----------------+
2 | dept_name       |
3 +-----------------+
4 | Training        |
5 | Design          |
6 | Implementation  |
7 +-----------------+
8 3 rows in set (0.00 sec)
```

Duplicates

- In SQL, results may have identical rows
- Duplicates can be removed using the keyword DISTINCT

```
SELECT title FROM employees;
```

```
+-------------+
| title       |
+-------------+
| Trainer     |
| Technician  |
| Manager     |
| Designer    |
| Architect   |
| Manager     |
| Designer    |
| Manager     |
| Trainer     |
+-------------+
9 rows in set (0.00 sec)
```

Duplicates

- In SQL, results may have identical rows
- Duplicates can be removed using the keyword DISTINCT

```
1 SELECT DISTINCT title FROM employees;
```

```
1  +--------------+
2  | title        |
3  +--------------+
4  | Trainer      |
5  | Technician   |
6  | Manager      |
7  | Designer     |
8  | Architect    |
9  +--------------+
10 5 rows in set (0.00 sec)
```