Databases and Info Systems

Structured Query Language (SQL)

Dr. Seán Russell
sean.russell@ucd.ie,

School of Computer Science,
University College Dublin

March 4, 2020

Table of Contents

lecturers

lec_id	name
1	Sean Russell
2	David Lillis
3	Catherine Mooney
4	Shen Wang
5	Ruhai Dong
6	Brett Becker
7	John Dunnion

modules

code	module_name	lec_id
COMP1005J	Programming 2	7
COMP2001J	Computer Networks	4
COMP1002J	Program Construction 2	6
COMP2004J	Databases	5
COMP2007J	Computer Organisation	4
COMP2013J	Databases	1
COMP2003J	Data Structures 2	8
MATH4023J	Really Hard Maths	15

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

Table of Contents

Joining Tables

- SQL-2 introduced a new syntax for joins
- Joins are represented explicitly in the FROM clause

```
SELECT attr_expr [AS alias] {, attr_expr [AS
    alias] } FROM table [AS alias] { [JoinType]
    JOIN table [AS alias] ON JoinConditions } [
    WHERE OtherCondition ]
```

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

Join Types

- It is useful to learn the Cartesian product type of join (that we have seen before) when you start learning databases
- But generally they are not widely used.
- Instead, we normally use one of the following types of join:
  - INNER
  - RIGHT [OUTER]
  - LEFT [OUTER]
  - FULL [OUTER]

Inner Join

- Inner Join joins two tables together based on some condition
- Results are only returned for rows where there is a matching result in both tables
- If there is no match for a row in either table, then the row is not shown

Inner Join Example

- Query: Find all of the lecturers and the modules that they teach
- SQL:

```
SELECT * FROM lecturers AS l INNER JOIN modules AS m ON
    l.lec_id = m.lec_id;
```

- Result:

```
+--------+-----------------+---------+-------------------------+--------+
| lec_id | name            | code    | module_name             | lec_id |
+--------+-----------------+---------+-------------------------+--------+
|      6 | Brett Becker    | COMP1002J | Program Construction 2 |      6 |
|      7 | John Dunnion    | COMP1005J | Programming 2          |      7 |
|      4 | Shen Wang       | COMP2001J | Computer Networks      |      4 |
|      9 | Henry Mcloughlin | COMP2002J | Data Structures 1     |      9 |
|      5 | Ruhai Dong      | COMP2004J | Databases              |      5 |
|      4 | Shen Wang       | COMP2007J | Computer Organisation  |      4 |
|      1 | Sean Russell    | COMP2013J | Databases              |      1 |
+--------+-----------------+---------+-------------------------+--------+
7 rows in set (0.00 sec)
```

Left Join

- Left Join joins two tables together based on some condition
- Results are only returned for **every row** in the table on the **left** of the join
- If there is no match for a row in the right table, then the columns all show the value **NULL**

Left Join Example

- Query: Find the lecturers and their modules, including the lecturers that don't teach modules
- SQL:

```
SELECT * FROM lecturers AS l LEFT JOIN modules AS m ON l.lec_id
    = m.lec_id;
```

- Result:

```
+--------+-----------------+---------+-----------------------+--------+
| lec_id | name            | code    | module_name           | lec_id |
+--------+-----------------+---------+-----------------------+--------+
|      6 | Brett Becker    | COMP1002J | Program Construction 2 |      6 |
|      7 | John Dunnion    | COMP1005J | Programming 2         |      7 |
|      4 | Shen Wang       | COMP2001J | Computer Networks     |      4 |
|      9 | Henry Mcloughlin | COMP2002J | Data Structures 1     |      9 |
|      5 | Ruhai Dong      | COMP2004J | Databases             |      5 |
|      4 | Shen Wang       | COMP2007J | Computer Organisation |      4 |
|      1 | Sean Russell    | COMP2013J | Databases             |      1 |
|      2 | David Lillis    | NULL    | NULL                  |   NULL |
|      3 | Catherine Mooney | NULL    | NULL                  |   NULL |
|      8 | Takfarinas Saber | NULL    | NULL                  |   NULL |
+--------+-----------------+---------+-----------------------+--------+
10 rows in set (0.00 sec)
```

Right Join

- Join two tables together based on some condition
- Results are returned for **every row** in the table on the **right** of the join
- If there is no match for a row in the left table, then the columns all show the value **NULL**

Right Join Example

- Query: Find the lecturers and modules, including the modules that do not have a lecturer in the database
- SQL:

```
SELECT * FROM lecturers AS l RIGHT JOIN modules AS m ON
    l.lec_id = m.lec_id;
```

- Result:

```
+--------+---------------+---------+-----------------------+--------+
| lec_id | name          | code    | module_name           | lec_id |
+--------+---------------+---------+-----------------------+--------+
|      6 | Brett Becker  | COMP1002J | Program Construction 2 |      6 |
|      7 | John Dunnion  | COMP1005J | Programming 2          |      7 |
|      4 | Shen Wang     | COMP2001J | Computer Networks      |      4 |
|      9 | Henry Mcloughlin | COMP2002J | Data Structures 1   |      9 |
|      5 | Ruhai Dong    | COMP2004J | Databases              |      5 |
|      4 | Shen Wang     | COMP2007J | Computer Organisation  |      4 |
|      1 | Sean Russell  | COMP2013J | Databases              |      1 |
|   NULL | NULL          | MATH4023J | Really Hard Maths      |     15 |
+--------+---------------+---------+-----------------------+--------+
8 rows in set (0.00 sec)
```

Full Join

- Join two tables together based on some condition
- Results are returned for **every row** in the **both tables**
- If there is no match for a row in the left table, then the columns all show the value **NULL**
- If there is no match for a row in the right table, then the columns all show the value **NULL**

Right Join Example

- Query: Find the lecturers and modules, including the modules that do not have a lecturer in the database
- SQL:

```
SELECT * FROM lecturers AS l FULL JOIN modules AS m ON l.lec_id
    = m.lec_id;
```

- Result:

```
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that
    corresponds to your MySQL server version for the right syntax to use near
    'FULL JOIN modules AS m ON l.lec_id = m.lec_id' at line 1
```

- Full join is not supported in MySQL...

Alternative Full Join

- We can implement a full join, by performing 2 queries and putting the results together
- A left join and a right join put together using a UNION
- SQL:

```
SELECT l.lec_id, name, code, module_name FROM lecturers AS l LEFT
    JOIN modules AS m ON l.lec_id = m.lec_id UNION SELECT
    m.lec_id, name, code, module_name FROM lecturers AS l RIGHT
    JOIN modules as m ON l.lec_id = m.lec_id;
```

Alternative Full Join

- We can implement a full join, by performing 2 queries and putting the results together
- A left join and a right join put together using a UNION
- Result:

lec_id	name	code	module_name
6	Brett Becker	COMP1002J	Program Construction 2
7	John Dunnion	COMP1005J	Programming 2
4	Shen Wang	COMP2001J	Computer Networks
9	Henry Mcloughlin	COMP2002J	Data Structures 1
5	Ruhai Dong	COMP2004J	Databases
4	Shen Wang	COMP2007J	Computer Organisation
1	Sean Russell	COMP2013J	Databases
2	David Lillis	NULL	NULL
3	Catherine Mooney	NULL	NULL
8	Takfarinas Saber	NULL	NULL
15	NULL	MATH4023J	Really Hard Maths

11 rows in set (0.00 sec)

LEFT JOIN to find missing relationships

- LEFT JOIN will return every row in the left table
- If it does not match anything in the right table, these attributes will contain NULL values
- Find the names of all lecturers who do not teach any module
- SQL:

```
SELECT name FROM lecturers AS l LEFT JOIN modules as m ON
    l.lec_id = m.lec_id WHERE code IS NULL;
```

- Result:

```
+-----------------+
| name            |
+-----------------+
| David Lillis    |
| Catherine Mooney |
| Takfarinas Saber |
+-----------------+
3 rows in set (0.00 sec)
```

USING

- Joining tables is a very common operation in SQL
- When computer scientists do something a lot, we find ways to simplify it
- Many joins between two tables are that contain an attribute with the same name that we are comparing to find where they are the same
- In this situation we can use the keyword USING
- Syntax:

```
table [JoinType] JOIN table USING(attribute)
```

Using Examples

- Inner Join:

```
SELECT * FROM lecturers INNER JOIN modules USING(lec_id);
```

- Left Join:

```
SELECT * FROM lecturers LEFT JOIN modules USING(lec_id);
```

- Right Join:

```
SELECT * FROM lecturers RIGHT JOIN modules USING(lec_id);
```

Natural Join

- Natural Join takes this logic of saving time a little further
- Using a natural join, we perform an inner join using any attributes with the same name in both tables
- In our examples, this would be on the column `lec_id`, but it could be others
- Syntax:

```
table NATURAL JOIN table
```

Natural Example

- Inner Join:

```
SELECT * FROM lecturers NATURAL JOIN modules;
```

- Result:

```
+--------+-----------------+----------+-----------------------+
| lec_id | name            | code     | module_name           |
+--------+-----------------+----------+-----------------------+
|      6 | Brett Becker    | COMP1002J | Program Construction 2 |
|      7 | John Dunnion    | COMP1005J | Programming 2          |
|      4 | Shen Wang       | COMP2001J | Computer Networks      |
|      9 | Henry Mcloughlin | COMP2002J | Data Structures 1      |
|      5 | Ruhai Dong      | COMP2004J | Databases              |
|      4 | Shen Wang       | COMP2007J | Computer Organisation  |
|      1 | Sean Russell    | COMP2013J | Databases              |
+--------+-----------------+----------+-----------------------+
7 rows in set (0.00 sec)
```

Table of Contents

Ordering Data

- To order our data in a particular way, we use the `ORDER BY` clause
- Example

```
SELECT name, salary FROM employees ORDER BY salary;
```

  - Employees are ordered by salary
  - The default behaviour is to sort in ascending order (i.e. from lowest to highest).
  - You can reverse the sort by using the `DESC` keyword (short for **descending**)

Ordering Example

- Example

```
SELECT name, salary FROM employees ORDER BY salary;
```

- Result

```
+---------------------+--------+
| name                | salary |
+---------------------+--------+
| Denis Hickey        |  20000 |
| Brendan Macken      |  25000 |
| Peter Stringer      |  25000 |
| Ronan O'Gara        |  29000 |
| Brian O'Driscoll    |  45000 |
| Leo Cullen          |  45000 |
| Jamie Heaslip       |  47000 |
| Sean Russell        |  50000 |
| Sean O'Brien        |  50000 |
+---------------------+--------+
9 rows in set (0.00 sec)
```

Ordering Example

- Example

```
SELECT name, salary FROM employees ORDER BY salary DESC;
```

- Result

```
+--------------------+--------+
| name               | salary |
+--------------------+--------+
| Sean Russell       |  50000 |
| Sean O'Brien       |  50000 |
| Jamie Heaslip      |  47000 |
| Brian O'Driscoll   |  45000 |
| Leo Cullen         |  45000 |
| Ronan O'Gara       |  29000 |
| Brendan Macken     |  25000 |
| Peter Stringer     |  25000 |
| Denis Hickey       |  20000 |
+--------------------+--------+
9 rows in set (0.00 sec)
```

More Ordering

- Some employees in the database have equal salaries. You can ask these to be sorted by something else

```
SELECT name, salary FROM employees ORDER BY salary DESC,
    name;
```

  - Employees are ordered by salary (descending)
  - If two employees have the same salary, these will be ordered by their name.
  - Salary is ordered in descending order, but name in ascending order

Ordering Example

- Example

```
SELECT name, salary FROM employees ORDER BY salary DESC,
    name;
```

- Result

```
+--------------------+--------+
| name               | salary |
+--------------------+--------+
| Sean O'Brien       |  50000 |
| Sean Russell       |  50000 |
| Jamie Heaslip      |  47000 |
| Brian O'Driscoll   |  45000 |
| Leo Cullen         |  45000 |
| Ronan O'Gara       |  29000 |
| Brendan Macken     |  25000 |
| Peter Stringer     |  25000 |
| Denis Hickey       |  20000 |
+--------------------+--------+
9 rows in set (0.00 sec)
```

- We can use the `LIMIT` keyword to reduce the number of rows returned by a `SELECT` query
- This is useful for:
  - Exploring a large new database (you may not need to see all of the data)
  - Programs that display part of the data to users (there is no need to get all the data in a query)
- Syntax:

```
SELECT * FROM employees LIMIT 5;
```

Multiple Arguments

- We can give two arguments to LIMIT
  - The first is the starting index
  - The second is the maximum number of rows
- First 5 rows:

```
SELECT name, salary FROM employees ORDER BY salary, name
    LIMIT 0,5;
```

- Next 5 rows:

```
SELECT name, salary FROM employees ORDER BY salary, name
    LIMIT 5,5;
```

```
SELECT name, salary FROM employees ORDER BY salary, name LIMIT 0,5;
```

```
+--------------------+--------+
| name               | salary |
+--------------------+--------+
| Denis Hickey       |  20000 |
| Brendan Macken     |  25000 |
| Peter Stringer     |  25000 |
| Ronan O'Gara       |  29000 |
| Brian O'Driscoll   |  45000 |
+--------------------+--------+
5 rows in set (0.00 sec)
```

```
SELECT name, salary FROM employees ORDER BY salary, name LIMIT 5,5;
```

```
+----------------+--------+
| name           | salary |
+----------------+--------+
| Leo Cullen     |  45000 |
| Jamie Heaslip  |  47000 |
| Sean O'Brien   |  50000 |
| Sean Russell   |  50000 |
+----------------+--------+
4 rows in set (0.00 sec)
```

Table of Contents

Aggregate Queries

- An aggregate query performs some calculation on multiple rows
- SQL-2 has 5 aggregate operators
  - `COUNT` - count the number of rows
  - `SUM` - calculate the sum of an attribute for a set of rows
  - `MAX` - find the maximum value of an attribute for a set of rows
  - `MIN` - find the minimum value of an attribute for a set of rows
  - `AVG` - calculate the average of an attribute for a set of rows

Count

- COUNT returns the number of rows or distinct values;
- It can count the number of rows

```
SELECT COUNT(*) FROM departments;
```

```
+----------+
| COUNT(*) |
+----------+
|        4 |
+----------+
1 row in set (0.00 sec)
```

Count

- COUNT returns the number of rows or distinct values;
- Find the number of values in a particular column (NULL values are not counted)

```
SELECT COUNT(manager_id) FROM departments;
```

```
+--------------------+
| COUNT(manager_id)  |
+--------------------+
|                  3 |
+--------------------+
1 row in set (0.00 sec)
```

Count

- COUNT returns the number of rows or distinct values;
- Find the number of different values in a particular column (duplicates values are not counted)

```
SELECT COUNT(DISTINCT division) FROM departments;
```

```
+-----------------------------+
| COUNT(DISTINCT division) |
+-----------------------------+
|                           2 |
+-----------------------------+
1 row in set (0.00 sec)
```

Sum

- The SUM function returns the sum of a set of value (ignores NULL values)
- Example: Find the total cost of all employee salaries
- SQL:

```
SELECT SUM(salary) FROM employees;
```

- Result

```
+-------------+
| SUM(salary) |
+-------------+
|      336000 |
+-------------+
1 row in set (0.00 sec)
```

Max

- The MAX function returns the maximum value in a set of value
- Example: Find the highest salary of all employee
- SQL:

```
SELECT MAX(salary) FROM employees;
```

- Result

```
+-------------+
| MAX(salary) |
+-------------+
|       50000 |
+-------------+
1 row in set (0.00 sec)
```

Min

- The MIN function returns the minimum value in a set of value
- Example: Find the lowest salary of all employee
- SQL:

```
SELECT MIN(salary) FROM employees;
```

- Result

```
+-------------+
| MIN(salary) |
+-------------+
|       20000 |
+-------------+
1 row in set (0.00 sec)
```

Average

- The AVG function returns the average value of a set of values (ignores NULL values).
- Example: Find the average salary for all employee
- SQL:

```
SELECT AVG(salary) FROM employees;
```

- Result

```
+---------------+
| AVG(salary)   |
+---------------+
|  37333.3333   |
+---------------+
1 row in set (0.00 sec)
```

More Complex Aggregate Queries

- Aggregate query with restriction
- Find the highest paid employee who is a manager

```
SELECT MAX(salary) FROM employees WHERE title="Manager";
```

- Result:

```
+-------------+
| MAX(salary) |
+-------------+
|       47000 |
+-------------+
1 row in set (0.00 sec)
```

More Complex Aggregate Queries

- Aggregate query with join
- Find the average salary of all employees in the design department

```
SELECT AVG(salary) FROM employees INNER JOIN departments
    USING(dept_id) WHERE dept_name="Design";
```

- Result:

```
+-------------+
| AVG(salary) |
+-------------+
|  40000.0000 |
+-------------+
1 row in set (0.00 sec)
```

Aggregate Queries and Target List

- Attributes in a query must have same number of results
- You can't mix aggregated and non-aggregated attributes
  - unless you use `GROUP BY`, which we will discuss later
- Example:

```
SELECT name, MAX(salary) FROM employees;
```

- Result:

```
ERROR 1140 (42000): In aggregated query without GROUP BY, expression #1 of
     SELECT list contains nonaggregated column 'week5.employees.name'; this is
     incompatible with sql_mode=only_full_group_by
```

Aggregate Queries and Target List

- We can have queries using multiple aggregate functions
- They will all have the same number or results (1)
- Example:

```
SELECT MAX(salary), MIN(salary), AVG(salary) FROM employees;
```

- Result:

```
+-------------+-------------+-------------+
| MAX(salary) | MIN(salary) | AVG(salary) |
+-------------+-------------+-------------+
|       50000 |       20000 |  37333.3333 |
+-------------+-------------+-------------+
1 row in set (0.00 sec)
```

Table of Contents

- Queries may apply aggregate functions to only a **subset of rows**
- This allows us to use aggregate and non-aggregate attributes together
  - non-aggregate attributes must be in the `GROUP BY` clause
- For example, we might want to find the sum of all salaries for each department
- SQL:

```
SELECT dept_name, SUM(salary) FROM employees NATURAL JOIN
    departments GROUP BY dept_name;
```

- Result:

```
+----------------+-------------+
| dept_name      | SUM(salary) |
+----------------+-------------+
| Training       |      142000 |
| Design         |      120000 |
| Implementation |       74000 |
+----------------+-------------+
3 rows in set (0.00 sec)
```

Group predicates

- When conditions are on the result of an aggregate operator, it is necessary to use the `HAVING` clause
- For example, we might want to Find the departments where the average salary is over 40000
- SQL:

```
SELECT dept_id, AVG(salary) FROM employees GROUP BY dept_id
    HAVING AVG(salary) > 40000;
```

- Result:

```
+---------+---------------+
| dept_id | AVG(salary)   |
+---------+---------------+
| 10      |  47333.3333   |
+---------+---------------+
1 row in set (0.00 sec)
```

How It Works

1. The query is executed without GROUP BY and without aggregate operators (the WHERE clause is applied at this stage)
2. The query result is divided in subsets with the same values for the attributes appearing after the group by clause
3. The aggregate operator is applied separately to each subset (the HAVING clause is applied at this stage)

WHERE or HAVING

- Only expressions containing **aggregate operators** should appear in the argument of the `HAVING` clause, because the `HAVING` clause is applied after aggregation takes place
- The `WHERE` clause is applied before the aggregation takes place
- For example, we might want to find the name of the departments where the average salary is over 40000
- SQL:

```
SELECT dept_name FROM employees INNER JOIN departments
    USING(dept_id) GROUP BY dept_name HAVING AVG(salary) >
    40000;
```

WHERE and HAVING

- WHERE and HAVING can be used together
- Find the names of all departments that have an average salary less than 40000 for its employees that are not managers
- SQL:

```
SELECT dept_name, AVG(salary)
 FROM departments INNER JOIN employees USING(dept_id)
 WHERE title!="Manager"
 GROUP BY dept_id
 HAVING AVG(salary) < 40000;
```

- Note: WHERE must be before GROUP BY

Example

- Average salary of all departments

```
SELECT dept_name, AVG(salary) FROM departments INNER JOIN
    employees USING(dept_id) GROUP BY dept_id;
```

```
+----------------+---------------+
| dept_name      | AVG(salary)   |
+----------------+---------------+
| Training       |  47333.3333   |
| Design         |  40000.0000   |
| Implementation |  24666.6667   |
+----------------+---------------+
3 rows in set (0.00 sec)
```

Example

- Average salary less than 40000 (for all employees)

```
SELECT dept_name, AVG(salary) FROM departments INNER JOIN
    employees USING(dept_id) GROUP BY dept_id HAVING
    AVG(salary) < 40000;
```

```
+-----------------+---------------+
| dept_name       | AVG(salary)   |
+-----------------+---------------+
| Implementation  |  24666.6667   |
+-----------------+---------------+
1 row in set (0.00 sec)
```

Example

- Average salary less than 40000 (non-managers only)

```
SELECT dept_name, AVG(salary) FROM departments INNER JOIN
    employees USING(dept_id) WHERE title!='Manager' GROUP BY
    dept_id HAVING AVG(salary) < 40000;
```

```
+----------------+---------------+
| dept_name      | AVG(salary)   |
+----------------+---------------+
| Design         |  37500.0000   |
| Implementation |  22500.0000   |
+----------------+---------------+
2 rows in set (0.00 sec)
```