Databases - Resources

Gordon Royle

School of Mathematics & Statistics
University of Western Australia

Databases

For decades, databases in general, and *relational databases*in particular have been a substantial and critical part of the world's computational infrastructure.

Therefore the subject of *relational databases* and the `SQL` *query language* is one of the most mature components of a CS degree.

- It is taught in almost every university
- There is a vast amount of DB-related material online

Lectures

This unit (`CITS1402`) covers a conventional range of the fundamental concepts of Databases, namely SQL, its theoretical underpinnings and the most important concepts.

The lectures and associated lecture notes:

- Define the *content* and scope of the unit
  Ideas, concepts and techniques not described at all in the lectures[1] *will not be part of the unit*
- Provide a "guided tour" of the unit, highlighting the overall structure and introducing each of the major concepts
  Lectures do not*, and* cannot *provide an exhaustive compendium of every possible use or variation of every command or concept*

[1] or the labs, or the project

Labs

The labs provide illustrative examples designed to entrench and reinforce the concepts introduced in lectures. In general the lab questions will

- Start with *routine examples* of the concept
  Basically "change the names" from lecture examples
- Proceed with *simple variants* of the concept
  The same idea in different ways, exploring the numerous ways `SQL` *has for to achieve the same end*
- End with *challenge questions* that involve *novel uses* of the concept that are only mentioned *obliquely* in lectures
  For example, the self-join *in Lab 2 introduces a new concept that, while it follows logically from the definitions of joins, requires a* conceptual leap

Internal resources

- Lectures, all recorded
- Lab Demonstrators
  Go to any lab, as all are under-full, and go to several labs if you need
- `help1402` is super-important for two reasons
  - Other students can often help more quickly than I can, they can provide multiple different points of view, and they can explain exactly how *they* understood the concept.
  - If there are masses of questions on some particular topic, then I can spend extra time reinforcing that topic, whereas if a topic attracts no questions, then there is no particular reason for me to revisit it.
- Weekly "no question is too basic" workshop/tutorials
  Starting soon, aimed primarily at students who feel they need extra help

External Resources

There are numerous external resources

- Jennifer Widom's *Coursera* videos on Databases
  These were the "lectures" last year in the experimental "flipped classroom" teaching mode
- Books
  - Database Systems : The Complete Book, (Garcia-Molina, Ullman, Widom)
    `http://infolab.stanford.edu/~ullman/dscb.html`
  - Database Management Systems, (Ramakrishnan, Gehrke)
    `http://pages.cs.wisc.edu/~dbbook/`
- The MySQL documentation
  `http://dev.mysql.com/doc/refman/5.7/en/index.html`
- . . . and of course, Google

The SELECT statement

The key statement in SQL is the SELECT statement, which has the following *general form*:

SELECT *columns*
FROM *tables*
WHERE *conditions*
GROUP BY *group columns*
HAVING *more conditions*
ORDER BY *sort columns*
LIMIT *number*

The words in CAPITALS are the *keywords*, while the *italicised* terms are to be specified by the user.

The SELECT statement

13.2.9 SELECT Syntax

[+/-]

13.2.9.1 SELECT ... INTO Syntax

13.2.9.2 JOIN Syntax

13.2.9.3 UNION Syntax

```
SELECT
    [ALL | DISTINCT | DISTINCTROW ]
      [HIGH_PRIORITY]
      [MAX_STATEMENT_TIME = N]
      [STRAIGHT_JOIN]
      [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
      [SQL_CACHE | SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]
    select_expr [, select_expr ...]
    [FROM table_references
      [PARTITION partition_list]
    [WHERE where_condition]
    [GROUP BY {col_name | expr | position}
      [ASC | DESC], ... [WITH ROLLUP]]
    [HAVING where_condition]
    [ORDER BY {col_name | expr | position}
      [ASC | DESC], ...]
    [LIMIT {[offset,] row_count | row_count OFFSET offset}]
    [PROCEDURE procedure_name(argument_list)]
    [INTO OUTFILE 'file_name'
        [CHARACTER SET charset_name]
        export_options
      | INTO DUMPFILE 'file_name'
      | INTO var_name [, var_name]]
    [FOR UPDATE | LOCK IN SHARE MODE]]
```

The various parts

Learning SQL will initially focus on learning how to use the `SELECT` keyword, and all of its supporting keywords.

- The lecture `SQL-1` described how the *selection of output columns* works
  `SELECT columns`
- The lecture `SQL-2` described how the *selection of tables works*
  `FROM tables WHERE conditions`
- The lecture `SQL-3` will describe how the *summary features* work
  `GROUP BY group columns`
- . . . and so on

The output columns

The first thing we considered was the *output columns*:

The word `SELECT` is a compulsory keyword, while *columns* is a comma-separated list of *expressions involving column names*, with each expression determining one column of the *output table*.

- The *name* of a column

```
SELECT Name FROM City;
```

- The names of *several* columns

```
SELECT Name, CountryCode FROM City;
```

- An *expression* involving columns

```
SELECT Population / SurfaceArea FROM Country;
SELECT Length(Name) FROM City;
```

- A *wildcard*

```
SELECT * FROM Country;
```

The overview

SQL works as a *row-processing machine*.

- For each row determined by the `FROM` table (or tables)
  - The named columns (or expressions) are extracted (or calculated)
  - The resulting tuple is output as one row of the answer

If the query contains a `WHERE` clause, then this defines certain *conditions* that determine *which rows* undergo this process — only those that *satisfy* the conditions will be processed.

Reinforced

This idea was then reinforced:

We need to modify this in two ways — just print the *names* and only for the rows corresponding to `CITS1402`.

```
SELECT name
FROM Student, Enrolled
WHERE id = sid
      AND uid = 'CITS1402';
+-------+
| name  |
+-------+
| Bob   |
```

More advanced use

When working with `JOINS` the column specification becomes more complicated, because the column names *originate from* more than one table and so the names *might clash*.

```
SELECT *
FROM Student JOIN Enrolled JOIN Unit
ON Student.id = sid AND Unit.id = uid;
+------+-------+------+----------+----------+-------------+
| id   | name  | sid  | uid      | id       | name        |
+------+-------+------+----------+----------+-------------+
| 1    | Amy   | 1    | CITS1401 | CITS1401 | Databases   |
| 2    | Bob   | 2    | CITS1401 | CITS1401 | Databases   |
| 4    | Emily | 4    | CITS1401 | CITS1401 | Databases   |
```

So we learned how to disambiguate by giving “the full name” of the column.

Relational Algebra integration

These practical examples were reinforced by relational algebra

In `SQL` the keyword `SELECT` is used to specify *which columns* to be output — this is what the *projection operator* π does in relational algebra.

In `SQL` the keyword `WHERE` is used to specify *which rows* are to be processed — this is what the *selection operator* σ does in relational algebra.

Purpose	In `SQL`	In rel. alg
Choose cols	`SELECT`	π
Choose rows	`WHERE`	σ

More relational algebra

Now consider the expression

$$\pi_{\texttt{id}}\,(\texttt{Student})$$

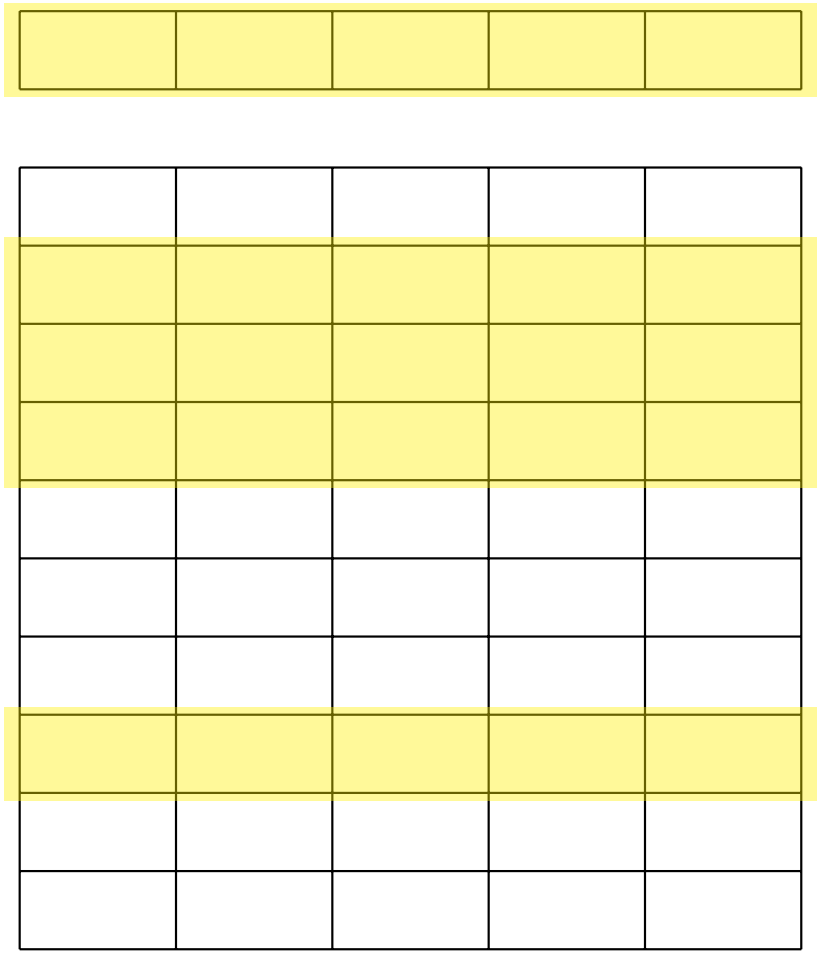
This goes through each row, and only keeps the *specified columns*.

In MySQL a *projection* is accomplished by explicitly listing the columns you want to keep.

```
SELECT id
FROM Student;
+-----------+
| id        |
+-----------+
| 12345678  |
| 12345682  |
```

Reminder - selection

The *select* operator σ selects *rows* of a table (inlcuding the header).



Reminder - Projection

The *project* operator π selects *columns* of a table, including the header.