# Lab Times

Labs will start on Thursday 1st October, there are two slots (9—11 and 16—18).

Lab groups are available on the website.

I am in the 9—11 group. Must find someone to swap with if possible.

# More SQL Basics (cont.)

SQL has a NOT operator, which negates a condition, and comes before it, e.g. (silly example):

```
WHERE NOT points >= 475;
```

If using with IN, could do WHERE hometown NOT IN(); or WHERE NOT hometown IN();.

## Reordering Results

You can control the appearance of the result table of your queries.
ORDER BY presents results in order of some attribute:

```
SELECT *
FROM students
ORDER BY points;
```

You can use ORDER BY points DESC will order from high to low instead of low to high.

Can also specify alternative column headings for greater readability:

```
SELECT
id_number AS 'Student ID',
points AS 'CAO Points'
FROM students
ORDER BY points;
```

Can use SQL functions to modify values appearing in results. For example, DATE_FORMAT(date_of_birth, '%W, %d %m %Y') lists dates as (e.g.): Monday, 13 June 1994

Also UPPER(last_name) puts last_name entries in uppercase.

There are ways of making your own functions, but it's not easy and we won't cover it in this course.

Missed a question here. Answer was that the uncertainty around the order things come back in (if not specified) comes from the complicated way databases work under the hood, where there are exotic implementations in order to make things efficient.

# SQL Data Manipulation Basics

SELECT statement extracts info but leaves the DB unchanged. Examples of manipulation statements are:

- INSERT: adds new rows to existing table
- DELETE: removes some rows from table
- UPDATE: changes some values within the table

These can destroy DBs if used incorrectly.

## INSERT

Example:

```
INSERT INTO students
VALUES ('987654321', 'Gráinne',…);
```

Not that values supplied must match the table columns in number, order, and type.
A value for every column must be specified.

## Inserting Multiple Rows

Example:

```
INSERT INTO students

VALUES

(…),

(…),

(…);
```

Note value sets are separated by commas and the last one must have a semicolon after it.

# Populating a DB from Scratch

You can use a bunch of INSERTs to set up an empty DB, and this can be done from a file, where the file contains multiple INSERT statements.

# Inserting Partial Rows

You can also use an INSERT statement and only supply some of the values. Missing values will be set to a default (typically NULL):

```
INSERT INTO students (id_number, first_name)
VALUES ('987654321', Gráinne);
```

# NULL

NULL is a special marker that is compatible with any domain/type. It's technically not a value.

Can be used in situations where a value isn't know, or is not applicable/irrelevant.

Over-reliance on NULLs may suggest poor DB.

# Bad Insertions

Insertions may be rejected for:

- key is a duplicate of an existing entry
- values are incompatible with column type etc.

Many bad insertions are legal according to SQL (e.g. mixing up columns or typing errors) and will not be rejected—these can cause lots of trouble.

Question: "What's a 'duplicate key'?"
Answer: "For every table, one column is specified as the key, and every value in that column must be unique. This ensures that every row has a unique identifier. So, you can't enter a value for the key column that is already taken."