

# Informatics 1: Data & Analysis

## Lecture 7: SQL

Ian Stark

School of Informatics  
The University of Edinburgh

Tuesday 7 February 2017  
Semester 2 Week 4



# Homework from Friday

## 1. Read This

Either or both of these study guides.



### Study Skill Essentials: Note Taking

The University of Manchester

<http://www.humanities.manchester.ac.uk/studyskills/essentials/note-taking/>



### How to Take Lecture Notes

wikiHow

<http://www.wikihow.com/Take-Lecture-Notes>

## 2. Do This

Work through the exercises for *Tutorial 2: Relational Modelling*. For more practice, try some of the additional examples.

## Data Representation

This first course section starts by presenting two common **data representation models**.

- The *entity-relationship (ER)* model
- The *relational* model

## Data Manipulation

This is followed by some methods for manipulating data in the relational model and using it to extract information.

- *Relational algebra*
- The *tuple relational calculus*
- The query language **SQL**

# SQL: Structured Query Language

- SQL is the standard language for interacting with relational database management systems
- Substantial parts of SQL are **declarative**: code states what should be done, not necessarily how to do it.
- When actually querying a large database, database systems take advantage of this to plan, rearrange, and optimize the execution of queries.
- Procedural parts of SQL do contain **imperative** code to make changes to the database.
- While SQL is an international standard (ISO 9075), individual implementations have notable idiosyncrasies, and code may not be entirely portable.

MySQL : PostgreSQL : Oracle : SQL Server : DB2 : SQLite : Sybase

# SQL Data Manipulation Language

In an earlier lecture we saw the SQL **Data Definition Language (DDL)**, used to declare the schema of relations and create new tables.

This lecture introduces the **Data Manipulation Language (DML)** which allows us to:

- Insert, delete and update rows in existing tables;
- Query the database.

Note that “query” here covers many different scales: from extracting a single statistic or a simple list, to building large tables that combine several others, or creating *views* on existing data.

SQL is a large and complex language. Here we shall only see some of the basic and most important parts. For a much more extensive coverage of the topic, sign up for the *Database Systems* course in Year 3.

## Inserting Data into a Table

```
CREATE TABLE Student (  
    uun    VARCHAR(8),  
    name  VARCHAR(20),  
    age   INTEGER,  
    email VARCHAR(25),  
    PRIMARY KEY (uun) )
```

The following adds a single record to this table:

```
INSERT  
    INTO Student (uun, name, age, email)  
    VALUES ('s1428751', 'Bob', 19, 'bob@sms.ed.ac.uk')
```

For multiple records, repeat; or consult your RDBMS manual.

Strictly, SQL allows omission of the field names; but if we include them, then the compiler will check them against the schema for us.

# Update and Delete Rows in a Table

## Update

This command changes the **name** recorded for one student:

```
UPDATE Student  
  SET name = 'Bobby'  
  WHERE uun = 's1428571'
```

## Delete

This deletes from the table all records for students named “Bobby”:

```
DELETE  
  FROM Students  
  WHERE name = 'Bobby'
```

# Simple Query

Extract all records for students older than 19.

```
SELECT *  
FROM Student  
WHERE age > 19
```

Returns a new table, with the same schema as **Student**, but containing only some of its rows.

Student

<b>uun</b>	<b>name</b>	<b>age</b>	<b>email</b>
s0456782	John	18	john@inf
s0378435	Helen	20	helen@phys
s0412375	Mary	18	mary@inf
s0189034	Peter	22	peter@math



# Simple Query

Extract all records for students older than 19.

```
SELECT *  
FROM Student  
WHERE age > 19
```

Returns a new table, with the same schema as **Student**, but containing only some of its rows.

Student

<b>uun</b>	<b>name</b>	<b>age</b>	<b>email</b>
s0456782	John	18	john@inf
s0378435	Helen	20	helen@phys
s0412375	Mary	18	mary@inf
s0189034	Peter	22	peter@math

# Simple Query

Extract all records for students older than 19.

```
SELECT *  
FROM Student  
WHERE age > 19
```

Returns a new table, with the same schema as `Student`, but containing only some of its rows.

<b>uun</b>	<b>name</b>	<b>age</b>	<b>email</b>
s0378435	Helen	20	helen@phys
s0189034	Peter	22	peter@math

# Simple Query

Extract all records for students older than 19.

```
SELECT *  
FROM Student  
WHERE age > 19
```

Returns a new table, with the same schema as `Student`, but containing only some of its rows.

## Tuple Relational Calculus

SQL is similar in form to the **comprehensions** of tuple relational calculus:

$$\{ S \mid S \in \text{Student} \wedge S.\text{age} > 19 \}$$

Working out how to implement this efficiently through relational algebra operations is the job of an SQL compiler and database query engine.

# Simple Query

Extract all records for students older than 19.

```
SELECT *  
FROM Student  
WHERE age > 19
```

Returns a new table, with the same schema as `Student`, but containing only some of its rows.

## Variations

We can explicitly name the selected fields.

```
SELECT uun, name, age, email  
FROM Student  
WHERE age > 19
```

# Simple Query

Extract all records for students older than 19.

```
SELECT *  
FROM Student  
WHERE age > 19
```

Returns a new table, with the same schema as `Student`, but containing only some of its rows.

## Variations

We can identify which table the fields are from.

```
SELECT Student.uun, Student.name, Student.age, Student.email  
FROM Student  
WHERE Student.age > 19
```

# Simple Query

Extract all records for students older than 19.

```
SELECT *  
FROM Student  
WHERE age > 19
```

Returns a new table, with the same schema as `Student`, but containing only some of its rows.

## Variations

We can locally abbreviate the table name with an *alias*.

```
SELECT S.uun, S.name, S.age, S.email  
FROM Student AS S  
WHERE S.age > 19
```

# Simple Query

Extract all records for students older than 19.

```
SELECT *  
FROM Student  
WHERE age > 19
```

Returns a new table, with the same schema as `Student`, but containing only some of its rows.

## Variations

We can save ourselves a very small amount of typing.

```
SELECT S.uun, S.name, S.age, S.email  
FROM Student S  
WHERE S.age > 19
```

# Anatomy of an SQL Query

```
SELECT field-list  
FROM table-list  
[ WHERE qualification ]
```

- The **SELECT** keyword starts the query.
- The list of fields specifies *projection*: what columns should be retained in the result. Using \* means all fields.
- The **FROM** clause lists one or more tables from which to take data.
- An optional **WHERE** clause specifies *selection*: which records to pick out and return from those tables.



# Anatomy of an SQL Query

```
SELECT field-list  
FROM table-list  
[ WHERE qualification ]
```

The *table-list* in the **FROM** clause is a comma-separated list of tables to be used in the query:

```
...  
FROM Student, Takes, Course  
...
```

Each table can be followed by an alias **Course AS C**, or even just **Course C**.

# Anatomy of an SQL Query

```
SELECT field-list  
FROM table-list  
[ WHERE qualification ]
```

The *field-list* after **SELECT** is a comma-separated list of expressions involving names of fields from the tables in **FROM**.

```
SELECT name, age
```

```
...
```

```
...
```

Field names can be referred to explicitly using table names or aliases: such as **Student.name** or **C.title**.

# Anatomy of an SQL Query

```
SELECT field-list  
FROM table-list  
[ WHERE qualification ]
```

The *qualification* in the **WHERE** clause is a logical expression built from tests involving field names, constants and arithmetic expressions.

...

...

```
WHERE age > 18 AND age < 65
```

Expressions can involve a range of numeric, string and date operations.

# Simple Query with Multiset Result

Extract all recorded student ages.

```
SELECT age  
FROM Student
```

Returns a new table, similar to `Student`, but containing only some of its columns.

Student

<b>uun</b>	<b>name</b>	<b>age</b>	<b>email</b>
s0456782	John	18	john@inf
s0378435	Helen	20	helen@phys
s0412375	Mary	18	mary@inf
s0189034	Peter	22	peter@math

# Simple Query with Multiset Result

Extract all recorded student ages.

```
SELECT age  
FROM Student
```

Returns a new table, similar to `Student`, but containing only some of its columns.

Student

uun	name	age	email
s0456782	John	18	john@inf
s0378435	Helen	20	helen@phys
s0412375	Mary	18	mary@inf
s0189034	Peter	22	peter@math

# Simple Query with Multiset Result

Extract all recorded student ages.

```
SELECT age  
FROM Student
```

Returns a new table, similar to `Student`, but containing only some of its columns.

<b>age</b>
18
20
18
22

## Aside: Multisets

The relational model given in earlier lectures has tables as *sets* of rows: so the ordering doesn't matter, and there are no duplicates.

Actual SQL does allow duplicate rows, with a **SELECT DISTINCT** operation to remove duplicates on request.

Thus SQL relations are not sets but *multisets* of rows. A multiset, or *bag*, is like a set but values can appear several times. The number of repetitions of a value is its *multiplicity* in the bag.

The following are distinct multisets:

$\{2, 3, 5\}$      $\{2, 3, 3, 5\}$      $\{2, 3, 3, 5, 5, 5\}$      $\{2, 2, 2, 3, 5\}$

Ordering still doesn't matter, so these are all the same multiset:

$\{2, 2, 3, 5\}$      $\{2, 3, 2, 5\}$      $\{5, 2, 3, 2\}$      $\{3, 2, 2, 5\}$

# Simple Query with Set Result

Extract the set of student ages in the table.

```
SELECT DISTINCT age  
FROM Student
```

Returns a new table, similar to `Student`, but containing only some elements from some of its columns.

Student

<b>uun</b>	<b>name</b>	<b>age</b>	<b>email</b>
s0456782	John	18	john@inf
s0378435	Helen	20	helen@phys
s0412375	Mary	18	mary@inf
s0189034	Peter	22	peter@math



# Simple Query with Set Result

Extract the set of student ages in the table.

```
SELECT DISTINCT age  
FROM Student
```

Returns a new table, similar to `Student`, but containing only some elements from some of its columns.

Student

uun	name	age	email
s0456782	John	18	john@inf
s0378435	Helen	20	helen@phys
s0412375	Mary	18	mary@inf
s0189034	Peter	22	peter@math

# Simple Query with Set Result

Extract the set of student ages in the table.

```
SELECT DISTINCT age  
FROM Student
```

Returns a new table, similar to `Student`, but containing only some elements from some of its columns.

<b>age</b>
18
20
22

# Quotation Marks in SQL Syntax

SQL uses alphanumeric **tokens** of three kinds:

- Keywords: **SELECT**, **FROM**, **UPDATE**, ...
- Identifiers: **Student**, **uun**, **age**, **S**, ...
- Strings: **'Bobby'**, **'Informatics 1'**, ...

Each of these kinds of token has different rules about **case sensitivity**, the use of **quotation marks**, and whether they can contain **spaces**.

While programmers can use a variety of formats, and SQL compilers should accept them, programs that *generate* SQL code are often rather cautious in what they emit and may use quotation everywhere possible.

Most SQL is written by machines, for machines.

# Know Your Syntax

		Case sensitive?	Spaces allowed?	Quotation character?	Quotation Required?
Keywords	<b>FROM</b>	No	Never	None	No
Identifiers	<b>Student</b>	Maybe	If quoted	"Double"	If spaces
Strings	'Bob'	It depends	Yes	'Single'	Always

For example:

```
select uun  
from Student as "Student Table"  
where "Student Table".age > 19 and "name" = 'Bobby Tables'
```

It's always safe to use only uppercase keywords and put quotation marks around all identifiers. Some tools will do this automatically.

Bring your solutions, or your attempts at them. You will need to be able to show these to your tutor and exchange them with other students.

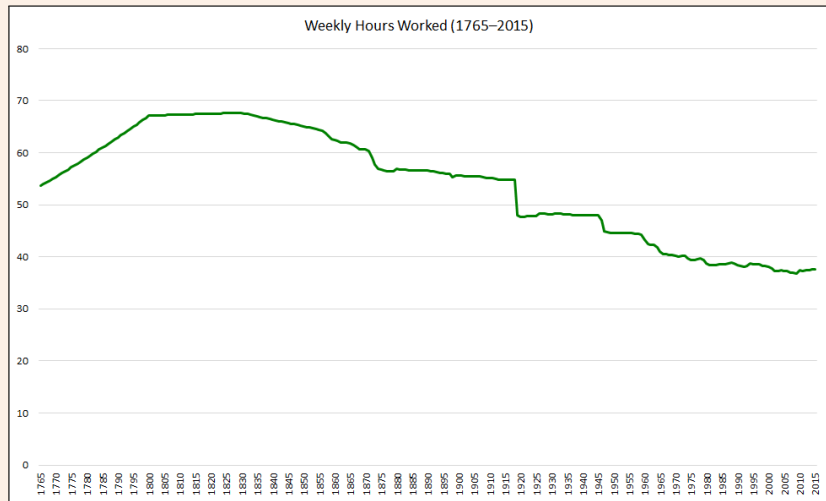
Come to tutorials prepared. Students who have not even attempted the exercises will be sent away to do them elsewhere and return later.

Even so, if you feel you are falling behind and cannot complete the work **do not skip the tutorial**. Instead, do what you can and then explain at the tutorial where you could make no more progress. Tutors are there to help you at whatever level you have reached.

You can also ask for help on [Piazza](#) at any time: there are six tutors and over 100 students there to give advice.

Most important: start early!

# Study Hours



Bank of England “Three Centuries of Macroeconomic Data”

- Full-time study is a full-time job. During all weeks of semester, plan to spend at least 40 hours each week across your courses.
- Within each week, balance time between courses according to their relative credit point weights out of 60.
- For a 10-point course like Inf1-DA, that suggests at least 6–7 hours of study per week.
- Inf1-DA has three contact hours per week, which means spending **at least the same again** in independent study.
- This rule of thumb applies to many courses: for every directly taught hour (lecture, tutorial, lab), add at least one hour for self-study.

Some activities are very different, such as field trips or research projects

Some students also undertake paid or voluntary work outside their studies. This can certainly be beneficial, and not just for the money: it widens your experience and is a chance get outside the university bubble.

However, too much outside work will affect your study and limit your achievements. I recommend the following guidelines:

- Never take on work that clashes with any timetabled class.
- Up to six hours a week is like taking an additional taught course: it's possible to do this, but requires you to be confident about your existing study load and committed to managing your time strictly.
- Beyond this begins to risk damaging your degree study. I suggest an absolute maximum of 12 hours a week during teaching times..

The University states a limit of 15 outside hours each week: giving a baseline of 55 hours working each week, which I think is unsustainable.

<http://www.ed.ac.uk/careers/looking-for-work/part-time-vacation/combining-work-study>



# Simple Query

Extract all records for students older than 19.

```
SELECT *  
FROM Student  
WHERE age > 19
```

Returns a new table, with the same schema as **Student**, but containing only some of its rows.

$$\{ S \mid S \in \text{Student} \wedge S.\text{age} > 19 \}$$

<b>uun</b>	<b>name</b>	<b>age</b>	<b>email</b>
s0378435	Helen	20	helen@phys
s0189034	Peter	22	peter@math

# Students and Courses

Student

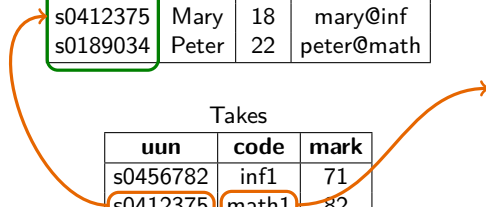
uun	name	age	email
s0456782	John	18	john@inf
s0378435	Helen	20	helen@phys
s0412375	Mary	18	mary@inf
s0189034	Peter	22	peter@math

Course

code	title	year
inf1	Informatics 1	1
math1	Mathematics 1	1
geo1	Geology 1	1
db3	Database Systems	3
adbs	Advanced Databases	4

Takes

uun	code	mark
s0456782	inf1	71
s0412375	math1	82
s0412375	geo1	64
s0189034	math1	56



## Example Query

Find the names and email addresses of all students taking Mathematics 1.

```
SELECT Student.name, Student.email
FROM Student, Takes, Course
WHERE Student.uun = Takes.uun
       AND Takes.code = Course.code
       AND Course.title = 'Mathematics 1'
```

Take rows from all three tables at once,

Student				Takes			Course		
uun	name	age	email	uun	code	mark	code	title	year
s0456780	John	18	john@inf	s0456780	inf1	71	inf1	Informatics 1	1
s0378435	Helen	20	helen@phys	s0412375	math1	82	math1	Mathematics 1	1
s0412375	Mary	18	mary@inf	s0412375	geo1	64	geo1	Geology 1	1
s0189034	Peter	22	peter@math	s0189034	math1	56	dbms	Database Systems	3
							adbs	Advanced Databases	4

## Example Query

Find the names and email addresses of all students taking Mathematics 1.

```
SELECT Student.name, Student.email
FROM Student, Takes, Course
WHERE Student.uun = Takes.uun
        AND Takes.code = Course.code
        AND Course.title = 'Mathematics 1'
```

Take rows from all three tables at once, pick out only those row combinations which match the test,

Student				Takes			Course		
uun	name	age	email	uun	code	mark	code	title	year
s0456780	John	18	john@inf	s0456780	inf1	71	inf1	Informatics 1	1
s0378435	Helen	20	helen@phys	s0412375	math1	82	math1	Mathematics 1	1
s0412375	Mary	18	mary@inf	s0412375	geo1	64	geo1	Geology 1	1
s0189034	Peter	22	peter@math	s0189034	math1	56	db1	Database Systems	3
							adbs	Advanced Databases	4

## Example Query

Find the names and email addresses of all students taking Mathematics 1.

```
SELECT Student.name, Student.email
FROM Student, Takes, Course
WHERE Student.uun = Takes.uun
        AND Takes.code = Course.code
        AND Course.title = 'Mathematics 1'
```

Take rows from all three tables at once, pick out only those row combinations which match the test,

Student				Takes			Course		
uun	name	age	email	uun	code	mark	code	title	year
s0456780	John	18	john@inf	s0456780	inf1	71	inf1	Informatics 1	1
s0378435	Helen	20	helen@phys	s0412375	math1	82	math1	Mathematics 1	1
s0412375	Mary	18	mary@inf	s0412375	geo1	64	geo1	Geology 1	1
s0189034	Peter	22	peter@math	s0189034	math1	56	dbms	Database Systems	3
							adbs	Advanced Databases	4

## Example Query

Find the names and email addresses of all students taking Mathematics 1.

```
SELECT Student.name, Student.email
FROM Student, Takes, Course
WHERE Student.uun = Takes.uun
        AND Takes.code = Course.code
        AND Course.title = 'Mathematics 1'
```

Take rows from all three tables at once, pick out only those row combinations which match the test, and return the named columns.

Student				Takes			Course		
uun	name	age	email	uun	code	mark	code	title	year
s0456780	John	18	john@inf	s0456780	inf1	71	inf1	Informatics 1	1
s0378435	Helen	20	helen@phys	s0412375	math1	82	math1	Mathematics 1	1
s0412375	Mary	18	mary@inf	s0412375	geo1	64	geo1	Geology 1	1
s0189034	Peter	22	peter@math	s0189034	math1	56	dbms	Database Systems	3
							adbs	Advanced Databases	4

## Example Query

Find the names and email addresses of all students taking Mathematics 1.

```
SELECT Student.name, Student.email
FROM Student, Takes, Course
WHERE Student.uun = Takes.uun
        AND Takes.code = Course.code
        AND Course.title = 'Mathematics 1'
```

Take rows from all three tables at once, pick out only those row combinations which match the test, and return the named columns.

name	email
Mary	mary@inf
Peter	peter@math

## Example Query

Find the names and email addresses of all students taking Mathematics 1.

```
SELECT Student.name, Student.email
FROM Student, Takes, Course
WHERE Student.uun = Takes.uun
      AND Takes.code = Course.code
      AND Course.title = 'Mathematics 1'
```

Take rows from all three tables at once, pick out only those row combinations which match the test, and return the named columns.

Expressed in tuple relational calculus:

$$\{ R \mid \exists S \in \text{Student}, T \in \text{Takes}, C \in \text{Course} . \\ R.\text{name} = S.\text{name} \wedge R.\text{email} = S.\text{email} \wedge S.\text{uun} = T.\text{uun} \\ \wedge T.\text{code} = C.\text{code} \wedge C.\text{title} = \text{"Mathematics 1"} \}$$



## Example Query

Find the names and email addresses of all students taking Mathematics 1.

```
SELECT Student.name, Student.email
FROM Student, Takes, Course
WHERE Student.uun = Takes.uun
        AND Takes.code = Course.code
        AND Course.title = 'Mathematics 1'
```

Take rows from all three tables at once, pick out only those row combinations which match the test, and return the named columns.

Implemented in relational algebra,

$$\pi_{\text{name,email}}(\sigma_{\text{Student.uun} = \text{Takes.uun}}(\text{Student} \times \text{Takes} \times \text{Course})) \\ \wedge \text{Takes.code} = \text{Course.code} \\ \wedge \text{Course.name} = \text{"Mathematics 1"}$$

## Example Query

Find the names and email addresses of all students taking Mathematics 1.

```
SELECT Student.name, Student.email  
FROM Student, Takes, Course  
WHERE Student.uun = Takes.uun  
        AND Takes.code = Course.code  
        AND Course.title = 'Mathematics 1'
```

Take rows from all three tables at once, pick out only those row combinations which match the test, and return the named columns.

Implemented in relational algebra, in several possible ways:

$$\pi_{\text{name,email}}(\sigma_{\text{title}=\text{"Mathematics 1"}}(\text{Student} \bowtie \text{Takes} \bowtie \text{Course}))$$
$$\pi_{\text{name,email}}(\text{Student} \bowtie (\text{Takes} \bowtie (\sigma_{\text{title}=\text{"Mathematics 1"}}(\text{Course}))))$$

# Query Evaluation

SQL **SELECT** queries are very close to a programming-language form for the expressions of the tuple relational calculus, describing the information desired but not dictating how it should be computed.

To do that computation, we need something more like relational algebra. A single **SELECT** statement combines the operations of join, selection and projection, which immediately suggests one strategy:

- Compute the complete cross product of all the **FROM** tables;
- Select all the rows which match the **WHERE** condition;
- Project out only the columns named on the **SELECT** line.

Crucially, real database engines don't do that. Instead, they use relational algebra to rewrite that procedure into a range of different possible *query plans*, estimate the cost of each — looking at indexes, table sizes, selectivity, potential parallelism — and then execute one of them.

Find the names and email addresses of all students taking Mathematics 1.

```
SELECT Student.name, Student.email  
FROM Student JOIN Takes ON Student.uun=Takes.uun  
           JOIN Course ON Takes.code = Course.code  
WHERE Course.title = 'Mathematics 1'
```

This is explicit **JOIN** syntax.

It has exactly the same effect as implicit **JOIN** syntax:

```
SELECT Student.name, Student.email  
FROM Student, Takes, Course  
WHERE Student.uun = Takes.uun  
       AND Takes.code = Course.code  
       AND Course.title = 'Mathematics 1'
```



## Homework (1/2): Do This

A *transaction* is a single coherent operation on a database. This might involve substantial amounts of data, or take considerable computation; but is intended to be an all-or-nothing action.

The features that characterise a reliable implementation of transactions are standardly initialized as the *ACID* properties.

### Task

Find out what each letter **A C I D** stands for here, and what those four terms mean.

# Homework (2/2): Try This

Try writing some SQL by hand using one of these web demonstrators.

Basic

[w3schools.com](https://w3schools.com)

<https://tinyurl.com/try-sql>

Includes a tutorial

[SQL Bolt](http://sqlbolt.com)

<http://sqlbolt.com>

Advanced

[SQL Fiddle](http://sqlfiddle.com)

<http://sqlfiddle.com>

The screenshot shows a web-based SQL query tool interface. At the top, it says "SQL Statement: Edit the SQL Statement, and click 'Run SQL' to see the result." Below this, the SQL query is entered: "SELECT \* FROM Products WHERE Price>75;". A "Run SQL »" button is visible. Below the button, the "Result:" section shows "Number of Records: 4" and a table with the following data:

ProductID	ProductName	SupplierID	CategoryID	Unit	Price
9	Mishi Kobe Niku	4	6	18 - 500 g pkgs.	97
20	Sir Rodney's Marmalade	8	3	30 gift boxes	81
29	Thüringer Rostbratwurst	12	6	50 bags x 30 saugs.	123.79
38	Côte de Blaye	18	1	12 - 75 cl bottles	263.5

On the right side of the interface, there is a "Your Database:" section with a help icon. It lists various tables and their record counts:

Tablename	Records
Customers	91
Categories	8
Employees	10
OrderDetails	518
Orders	196
Products	77
Shippers	3
Suppliers	29

Below this list is a "Restore Database" button.

# Summary

## SQL: Structured Query Language

A declarative language for interacting with relational databases. SQL provides facilities to define tables; to add, update, and remove tuples; and to query tables in complex ways.

## Writing Queries

Queries can be used to extract individual items of data or simple lists; to build large tables combining several others; and to generate *views* on these.

SQL queries take a standard form: **SELECT ... FROM ... WHERE ...** to identify the fields returned, the tables used, and which records to pick.

## Executing Queries

Database engines prepare multiple *query plans* and estimate their cost (in memory space, disk I/O, time) before choosing one to execute.