No lecture on Friday.

Our Running Example

This is a deliberately simple database for an example (single-table, only 6 students, not too many columns).

Note it's more convenient to split names into first_name and last_name fields, because then you can choose all the people of a certain first name, for example.

Dates are expressed YYYY-MM-DD in SQL, which is more convenient and for the moment we have to just take it for granted.

ID numbers are stored as strings rather than numbers because we never need to treat them as numbers and it's more convenient to treat them as strings.

Naming Conventions

SQL Rules:

- start with a letter
- composed of letters, digits, and underscores
- NB: no internal spaces etc. (e.g. first_name not first name)

Conventions:

- use lower-case letters for names
- use upper-case for keywords (e.g. SELECT)
- names should be concise but *meaningful* (e.g. id_number not y16id or id_of_student_in_question)

SQL Queries

An SQL query specifies what info we require from the database table(s).

The result returned will be an unnamed table.

We will look at SELECT queries first. These leave databases unchanged, even if screwed up.

SELECT-FROM Queries

Template:

```
SELECT list_of_attributes
FROM table-name;
```

list_of_attributes — list of columns of interest, comma- separated

table-name — specifies table

Each query is terminated with a semicolon.

Examples:

```
SELECT id_number
FROM students;
SELECT first_name, last_name
FROM students;
SELECT *
FROM students;
```

* specifies all columns.

Distinctness

Tuples in relations should be distinct, but the result table from

```
SELECT course
FROM students;
```

has duplicates.

This is a frustrating discrepancy between relation model and SQL implementations.

```
SELECT DISTINCT course
FROM students;
```

will remove duplicates.

Note that you shouldn't assume that results will come in any particular order unless specified. E.g. names won't automatically come out in alphabetical order.

SELECT-FROM-WHERE Queries

```
SELECT list_of_attributes

FROM table-name

WHERE condition;
```

Examples:

```
SELECT id_number

FROM students

WHERE points = 475;

SELECT first_name, last_name

FROM students

WHERE points >= 550;

SELECT first_name, last_name
FROM students
WHERE course = 'ck401';
```

Note that queries don't have to be formatted on separate lines, you can write it all in one line, but readability is hugely increased with separate lines.

Note also that ck401 is in quotes (single or double) because it is a string format value. You would also need to do this for ID numbers since those are stored as strings in this DB.

For each tuple X of table in turn, the WHERE condition is checked for each entry. For each tuple that satisfies the condition, the relevent entries are appended to the results table.

Technically this isn't how it implemented, because there are faster/more efficient ways, but it's a good conceptual picture.

Main SQL Operators

- =
- <> — not equal
- <

- <=
- >
- >=
- BETWEEN (includes both endpoints, first endpoint must be lower)

Conditions Involving Dates

```
SELECT first_name, last_name
FROM students
WHERE date_of_birth < '1980-01-01';
```

SQL treats dates as strings, for reasons we won't get into yet.

The format must be 'YYYY-MM-DD' or "YYYY-MM-DD". Should use single quotes for style and to be consistent.

Operators are relatively clear:

- *< '1980-01-01'* means "before 1 Jan. 1980"
- *BETWEEN '1980-01-01' AND '1980-12-31'* means "born during 1980" (includes both endpoints)