Two-Table Queries

Because SQL doesn't support sequences or arrays, we use a second table linked to the first to pull this off.

Favourite foods example

Use a table "persons" for the usual information, and then a second table "foods" (or whatever) where there are two columns: person_id and food, where each entry represents a favourite food for someone.

Then we have multiple entries for each person to capture the various favourite foods of each person.

Queries requiring info from both tables

E.g. list the favourite foods of the person named Aoife Ahern or list all the persons who like pizza and beer.

For both of these queries, you need to access info from both tables. In the first you need to get the id for Aoife Ahern from the first table.

Cross product

Is a way of taking two sets and combining them. Is the set of all ordered combinations of attributes from the sets.

See CS1112 for more info.

Cross product in SQL

Can use CROSS JOIN to form the cross product.

```
SELECT *
FROM persons CROSS JOIN favorite_foods
```

The number of rows in the result will be the product of the numbers of rows in both tables—this will be enormous! Every row from the first table will be matched with every row from the second table.

The number of columns in the result will be the sum of the numbers of columns in the two tables.

Most rows in the cross join are meaningless, i.e. the ones with one person's information combined with another person's food preference.

On the other hand, the ones where the person is the same in both sections are useful. These columns will have the same person_id in two places, which allows us to identify these rows, and which is what makes them useful.

Now we want to filter out the useless info.

Joins in SQL

```
SELECT *
FROM persons JOIN favourite_foods
ON persons.person_id = favourite_foods.person_id;
```

So this gives us a subset of the cross product of the two tables which includes only the rows where the person_id values (one from each table) match up.

Note the join is no longer called CROSS JOIN.

Note the "persons." prefix refers to things in the persons table. You can use it in a single-table context and it still works.

You can use WHERE instead of ON, but we're using ON here because it's more efficient (it works in a slightly different way, throwing away junk info earlier).

We can think of this in two ways

Either you think of combining the tables and then eliminating the junk rows, as we described it before,
Or you can think of it combining only the rows which have the same person_id value.

How it works under the hood may be very different in order to improve efficiency, but this is how we conceptualise it.

An example

List all the students who like pizza:

```
SELECT *
FROM persons JOIN favourite_foods
ON persons.person_id = favourite_foods.person_id
WHERE food = 'Pizza';
```

There are a couple of ways we could rephrase this. We could replace the ON condition with a combined WHERE:

```
…
WHERE persons.person_id = favourite_foods.person_id AND food
```

= 'Pizza';

A tidier version

```
SELECT first_name, last_name, ' likes ', food
FROM
persons AS p
JOIN favourite_foods AS f
ON p.person_id = f.person_id
WHERE food = 'pizza';
```

The third column will contain the fixed string ' likes ' in every row.

We also re-label persons as p and favourite_foods as f to make selection easier with p.person_id and f.person_id.
This improves readability because we can use "p" for the rest of the query instead of "persons".

Example 2

List all pairs of students who hail from the same county (not all those from one specific county.

We use just the persons table but join it with itself.

```
SELECT

p1.first_name, p1.last_name, ' and ',
p2.first_name, p2.last_name, ' both come from ',
p1.county

FROM

persons AS p1
JOIN persons AS p2
ON p1.county = p2.county;
```

There are a couple of problems here:

- We get duplicates (Aoife-Barry and Barry-Aoife)
- We get self-pairs (Aoife-Aoife)

Can refine it:

```
SELECT

p1.first_name, p1.last_name, ' and ',
```

```
p2.first_name, p2.last_name, ' both come from ',
p1.county

FROM

persons AS p1
JOIN persons AS p2
ON p1.county = p2.county

AND p1.person_id < p2.person_id;
```

The < condition removes duplicates *and* self-pairs.