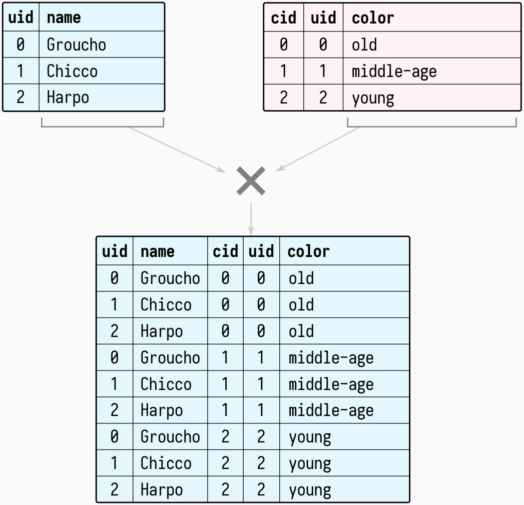
SELECT from multiple relations

If we use more than one table in a SELECT, the cartesian product is produced:

```
SELECT *
FROM r1, r2;
```



uid	name
0	Groucho
1	Chicco
2	Harpo

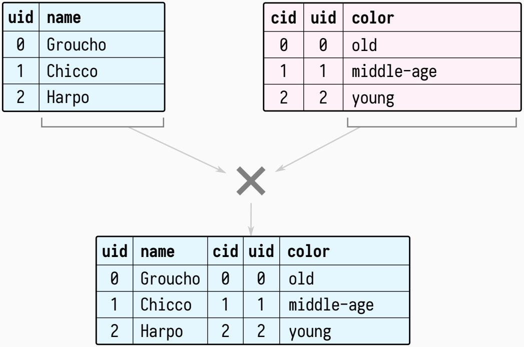
cid	uid	color
0	0	old
1	1	middle-age
2	2	young

uid	name	cid	uid	color
0	Groucho	0	0	old
1	Chicco	0	0	old
2	Harpo	0	0	old
0	Groucho	1	1	middle-age
1	Chicco	1	1	middle-age
2	Harpo	1	1	middle-age
0	Groucho	2	2	young
1	Chicco	2	2	young
2	Harpo	2	2	young

Filtering the Cartesian Product

To produce the desired result, we have to restrict the cartesian product:

```
SELECT *
FROM r1, r2
WHERE
  r2.uid = r1.uid;
```



uid	name
0	Groucho
1	Chicco
2	Harpo

cid	uid	color
0	0	old
1	1	middle-age
2	2	young

uid	name	cid	uid	color
0	Groucho	0	0	old
1	Chicco	1	1	middle-age
2	Harpo	2	2	young

More Examples

List Addresses with city name:

```sql
SELECT address.address, city.city
  FROM address, city
  WHERE address.city_id = city.city_id;
```

List all cities with country name:

```sql
SELECT city.city, country.country
  FROM city, country
  WHERE city.country_id = country.country_id;
```

Abbreviation with **AS**

We can assign an alias to tables to be able to refer to them more easily

```sql
SELECT a.address, c.city
  FROM address AS a, city AS c
  WHERE a.city_id = c.city_id;
```

The AS keyword is optional

```sql
SELECT a.address, c.city
  FROM address a, city c
  WHERE a.city_id = c.city_id;
```

NATURAL JOIN

The ”natural” join assumes columns with *the same name*.

```sql
SELECT a.city, b.country
  FROM city a NATURAL JOIN country b;
```

You can rename columns with `ALTER TABLE`:

```sql
ALTER TABLE users RENAME COLUMN id TO uid;
```

Problems

- Sometimes different columns have identical names and no relationship and a `NATURAL JOIN` is not possible.

JOIN ... USING

When a NATURAL JOIN is not possible, we can specify which columns to match with USING:

```
SELECT a.city, b.country
  FROM city a JOIN country b
  USING(country_id);
```

```
SELECT a.address, c.city
  FROM address a JOIN city c
  USING(city_id);
```

JOIN ... ON

If ID columns do not have identical names, with ON we can specify a general condition:

```sql
SELECT ci.city, co.country
  FROM city ci
  JOIN country co
  ON ci.country_id = co.id;
```

```sql
SELECT c.first_name, c.last_name, s.store_id
  FROM customer c
  JOIN store s
  ON c.store_id = s.id;
```

Joining 3 Tables

What cities do customers live in (hierarchical):

```sql
SELECT c.first_name, c.last_name, ci.city
  FROM customer c
  JOIN address a USING(address_id)
  JOIN city ci USING(city_id)
  ORDER BY ci.city;
```

Show each film with category (free interaction):

```sql
SELECT f.title, f.release_year, c.name as category
  FROM film f
  JOIN film_category USING(film_id)
  JOIN category c USING(category_id)
  ORDER BY f.title;
```

Joining 4 Tables

What movies did all customers rent

```
SELECT c.first_name, c.last_name, f.title, r.rental_date
  FROM customer c
  JOIN rental r USING(customer_id)
  JOIN inventory USING(inventory_id)
  JOIN film f USING(film_id);
```

Joining 6 Tables

What movies of ”Family” category did all customers rent:

```
SELECT c.first_name, c.last_name, r.rental_date, f.title
  FROM customer c
  JOIN rental r USING(customer_id)
  JOIN inventory USING(inventory_id)
  JOIN film f USING(film_id)
  JOIN film_category USING(film_id)
  JOIN category USING(category_id)
  WHERE category.name = 'Family'
  ORDER BY r.rental_date;
```

`INNER JOIN`s vs `OUTER JOIN`s

The `INNER JOIN` will show only the intersection between tuples in the cartesian product.

In a `SELECT` like

```sql
SELECT * from ta NATURAL JOIN tb;
```

there could be:

- Tuples in `ta` with no related tuples in `tb`,
- Tuples in `tb` with no related tuples in `ta`.

`OUTER JOIN`s just list this unpaired tuples.

LEFT JOIN

If we do a INNER JOIN for inventory, certain films will not appear because they do not have copies.

```sql
SELECT f.film_id, f.title, i.inventory_id
  FROM film f
  LEFT JOIN inventory i USING(film_id);
```

To specifically show films with no inventory:

```sql
SELECT f.film_id, f.title, i.inventory_id
  FROM film f
  LEFT JOIN inventory i USING(film_id)
  WHERE i.inventory_id IS NULL;
```