Advanced JOINS

Use the `spaces` table for this introduction. The needed tables were created in the *Normalization* session. If you didn't follow that lecture, you can create the neccessary tables using the hydenv CLI.

```
python -m hydenv examples spaces --normalize
```

Below, you will find the SQL queries used in the video, followed by a summary of the lessons learned.

SQL commands in the video

1:45:

```
-- recall the view
SELECT * FROM gps_missions_from_usa;
```

2:35:

```
-- implicit inner join
SELECT * FROM spacess
JOIN gps_missions_from_usa gps ON gps.company_id=spaces.company_id
```

4:12:

```
-- implicit inner join
SELECT * FROM spaces
INNER JOIN gps_missions_from_usa gps ON gps.company_id=spaces.company_id
```

5:27:

```
-- left join
SELECT * FROM spaces
LEFT JOIN gps_missions_from_usa gps ON gps.company_id=spaces.company_id
```

7:35:

```
-- right join
SELECT * FROM spaces
RIGHT JOIN gps_missions_from_usa gps ON gps.company_id=spaces.company_id
```

9:53:

```
SELECT * FROM spaces
RIGHT OUTER JOIN
(
  SELECT * FROM gps_missions_from_usa gps
  UNION
  (
    SELECT
        424242 AS company_id,
        9999 AS missions,
        '42 years' AS "serving years",
        'AAA' as company
    )
) gps
ON gps.company_id=spaces.company_id
ORDER BY company ASC
```

Summary

Consider again the JOIN from the lecture about working on relations:

customers:

customer_id	product_id	store_id
3	3948573	1
4	3902938	1
5	3948573	2

products:

products_id	name
3948573	socks
3902938	pullover

The join used was:

```
SELECT
  customer_id
FROM customers
JOIN products ON products.product_id=customers.products_id
WHERE name='socks'
```

Here, the **whole** products table is joined to the customers table. Then, the customers table is filtered by only the ones needed. As the customer table grows larger, it can be helpful to first filter the articles and then perform the join.

Inner Joins

The type of join we need here is an `INNER JOIN` to the filtered `products` table.

```
SELECT
  customer_id
FROM customers
```

```
INNER JOIN (SELECT * FROM products WHERE name='socks') AS sock_table
ON sock_table.product_id=customers.product_id
```

This can speed up the query by magnitudes.

Left / Right Joins

A join has a direction. You always take all datasets from one table and then join all the datasets from the other table that fulfill the `ON` clause.

Narrow down the products table by reducing it to the 10 most purchased products:

```
CREATE TEMPORARY VIEW topseller AS
SELECT p.product_id, p.name, count(c.customer_id) AS solds
FROM customers AS c
JOIN products AS p ON p.product_id=c.product_id
GROUP BY p.product_id, p.name
ORDER BY solds
LIMIT 10
```

Now, by joining this new *table* to the customers and counting the size, we see the different behavior demonstrated:

```
SELECT count(*) FROM customers
LEFT JOIN topseller ON topseller.product_id=customers.product_id
```

and:

```
SELECT count(*) FROM customers
RIGHT JOIN topseller ON topseller.product_id=customers.product_id
```