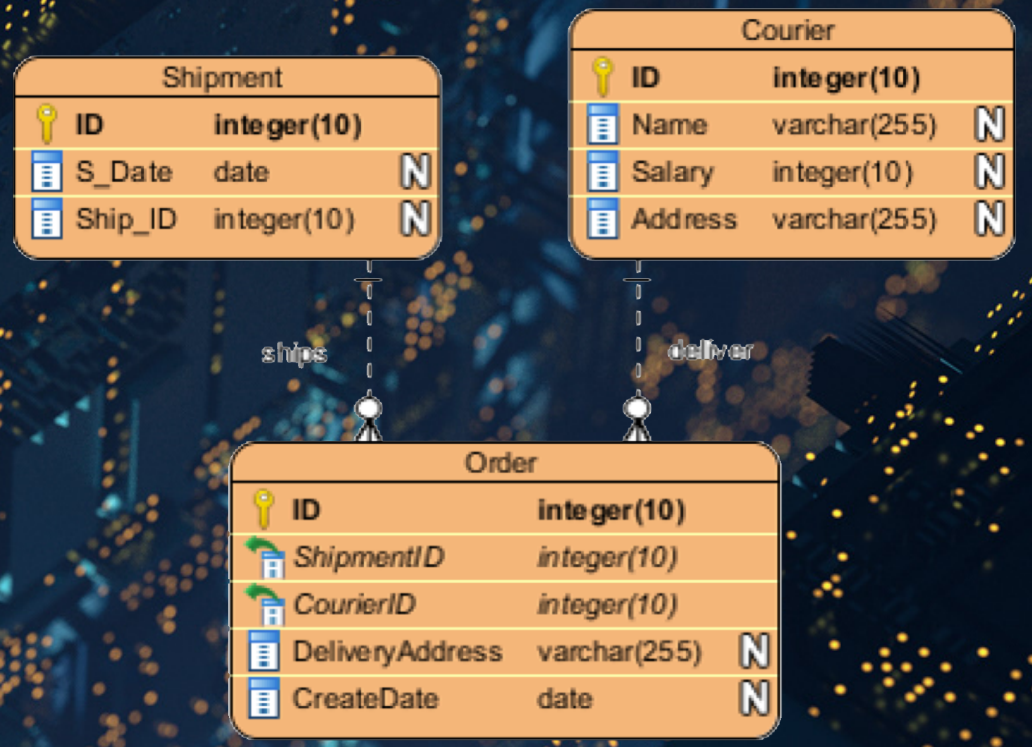

SQL - Joins

Contents

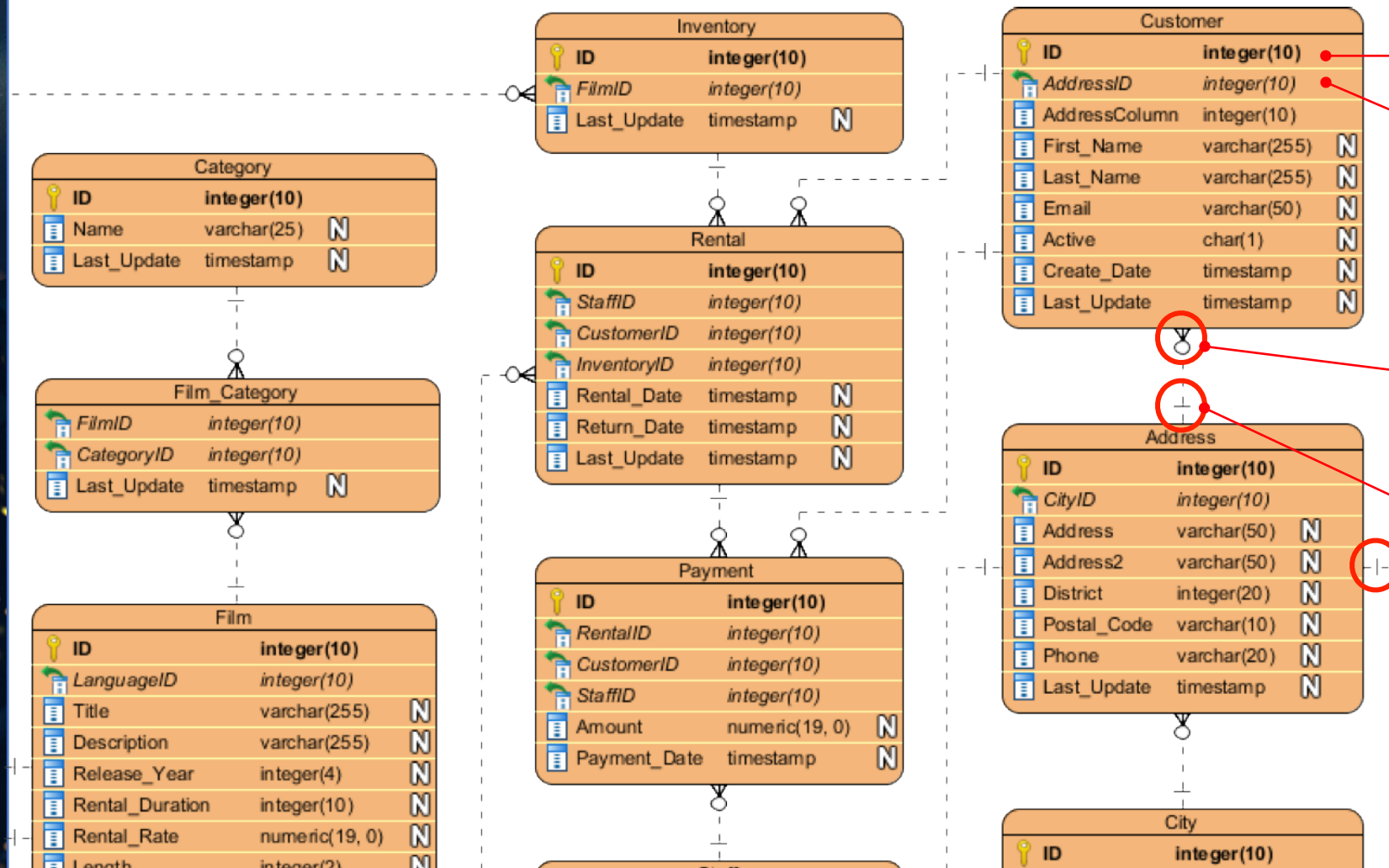
- ERD - Brief Review
- Why using `JOIN`?
- `JOIN` Statement
  - `INNER JOIN`
  - `LEFT JOIN`
  - `RIGHT JOIN`
  - `FULL OUTER JOIN`

Using multiple tables

- Take a look at our friend, the ERD:
- So far, we just used one entity
- Power of SQL comes from the fact we can run queries against multiple tables at once
- `JOINS` are the statement we use to 'connect' the tables together
- Joins work thanks to ***Normalization***, a design technique that reduces data redundancy



EDR Brief Review



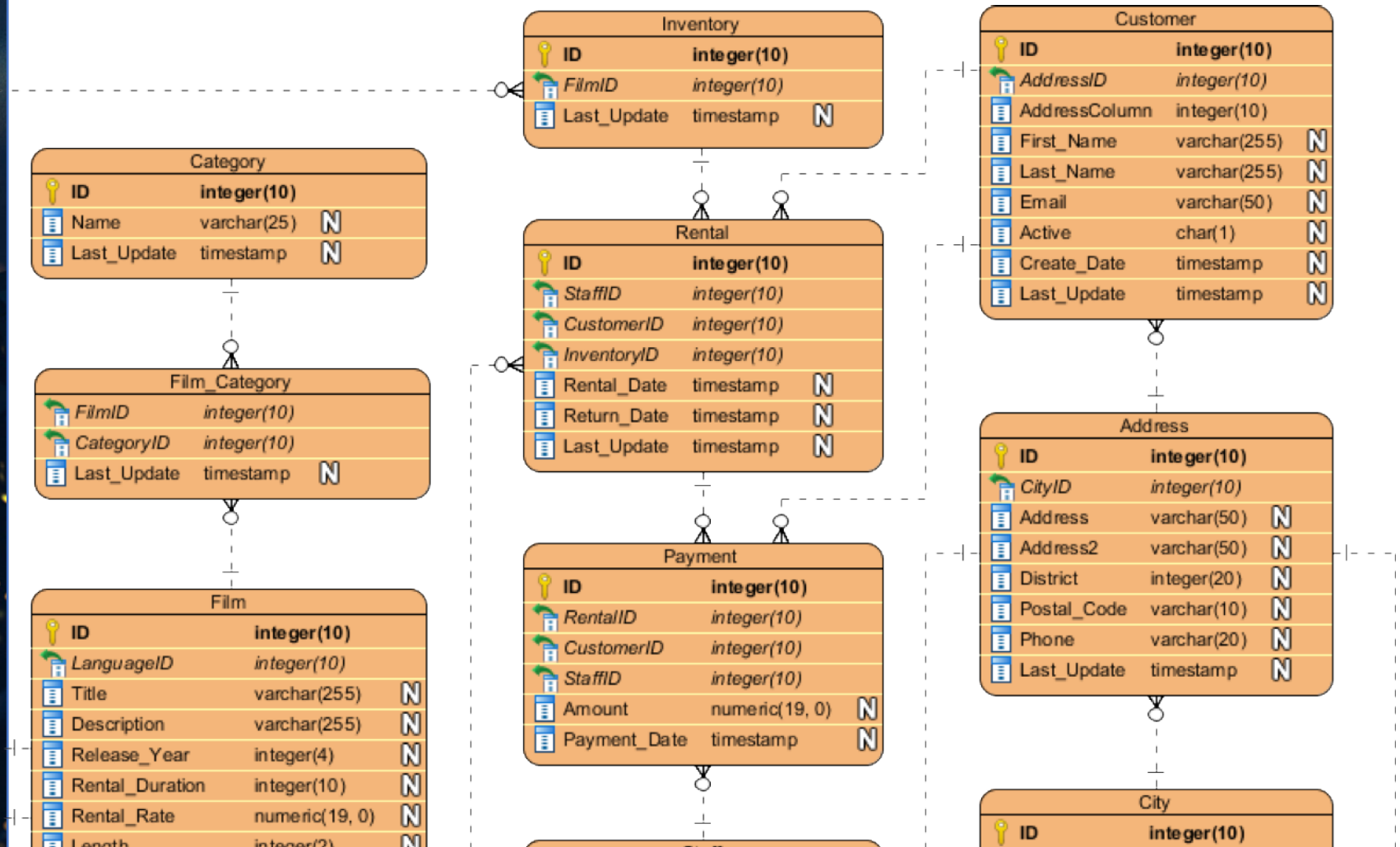
Primary Key (PK):
Unique identifiers

Foreign Key (FK):
Is a column in one table which is a PK

Crow's foot:
Indicates many of the instances of a **FK**

Dash line:
indicates only one instance of PK will be present in this table

EDR Brief Review

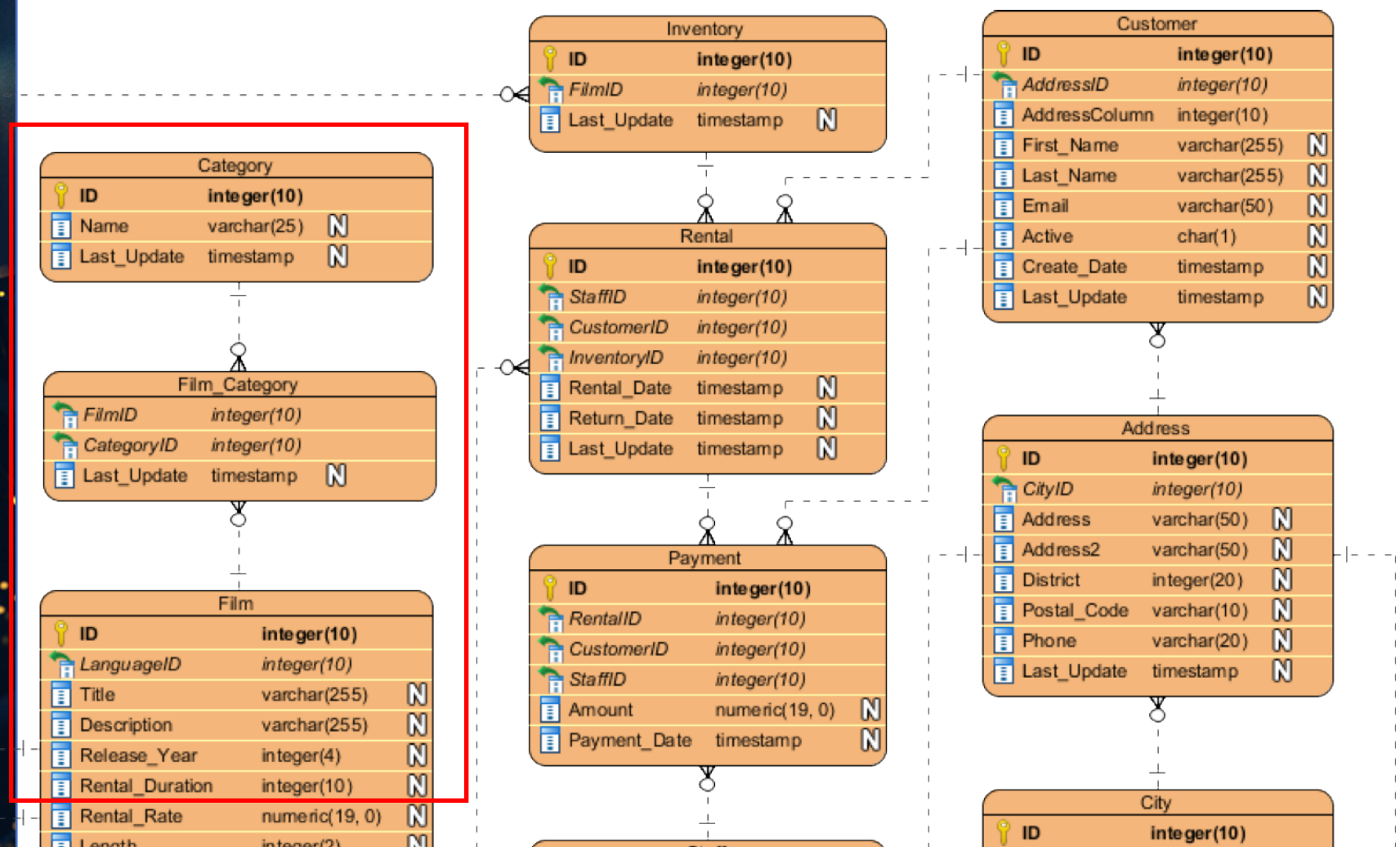


One to Many relationships

- Each Customer has a unique ID.
- ID it is the PK of the customer table
- Thus, we will find a single ID for each customer

- This ID is also present in the Rental, Payment
- In those tables, it is present as a FK
- Thus, we might find the same Customer ID for many rental operations.

EDR Brief Review



Many-to-Many relationships

- There are many categories
- There are many films
- One film can have many categories
- One category can be related to many films

- Since PKs have to be unique, many to many relationships have to resort to a intermediate step *(Film Category)*

EDR Brief Review

How can we leverage these relationships?

- As many tables are connected, we can perform interesting analyses
- We can see, for example, the stores in UK

Let's see this without using JOIN first

Why using JOIN

- First, let's find the UK id

```
SELECT * FROM country
WHERE country = 'United Kingdom';
```

	country_id [PK] integer	country character varying (50)	last_update timestamp without time zone
1	102	United Kingdom	2006-02-15 09:44:00

We can look cities whose country_id =102

Why using JOIN

```
SELECT * FROM city
WHERE country_id = 102;
```

	city_id [PK] integer	city character varying (50)	country_id smallint	last_update timestamp without time zone
1	88	Bradford	102	2006-02-15 09:45:25
2	149	Dundee	102	2006-02-15 09:45:25
3	312	London	102	2006-02-15 09:45:25
4	494	Southampton	102	2006-02-15 09:45:25
5	495	Southend-on-Sea	102	2006-02-15 09:45:25
6	496	Southport	102	2006-02-15 09:45:25
7	500	Stockport	102	2006-02-15 09:45:25
8	589	York	102	2006-02-15 09:45:25

Why using JOIN

```
SELECT * FROM address
WHERE city_id IN (88, 149, 312, 494, 495, 496, 500,
589);
```

	address_id [PK] integer	address character varying (50)	address2 character varying (50)	district character varying (20)	city_id smallint
1	20	360 Toulouse Parkway		England	495
2	89	1557 Ktahya Boulevard		England	88
3	146	483 Ljubertsy Parkway		Scotland	149
4	256	1497 Yuzhou Drive		England	312
5	482	808 Naala-Porto Parkway		England	500
6	502	1515 Korla Way		England	589
7	517	548 Uruapan Street		Ontario	312
8	562	869 Shikarpur Way		England	496
9	589	1584 Ljubertsy Lane		England	494

Why using JOIN

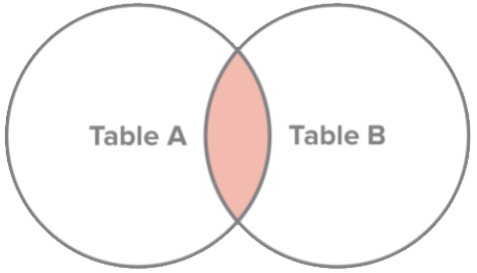
- The process is tedious
- We have to code the IDs manually
- We can’t guarantee that we got all the IDs

- JOIN makes this process simple
- There are four type of JOIN

Why using JOIN

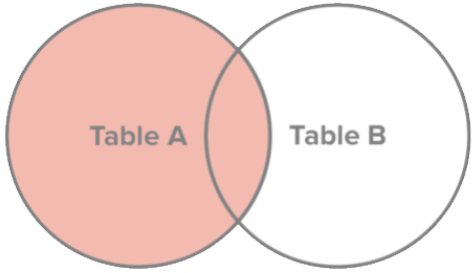
- There are four type of JOIN:

INNER JOIN



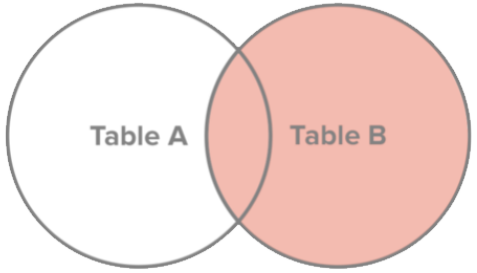
Select all records from Table A and Table B, where the join condition is met.

LEFT JOIN



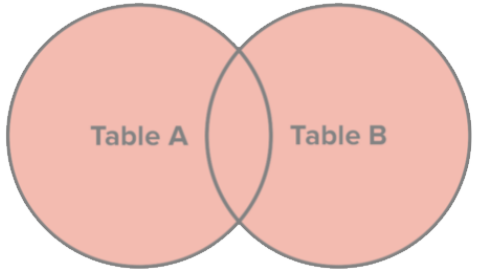
Select all records from Table A, along with records from Table B for which the join condition is met (if at all).

RIGHT JOIN



Select all records from Table B, along with records from Table A for which the join condition is met (if at all).

FULL OUTER JOIN



Select all records from Table A and Table B, regardless of whether the join condition is met or not.

JOIN Statement - INNER

- `INNER JOIN`: Connects rows based on a condition known as the ***join predicate***. It has the same functionality as `JOIN` (pure syntactic sugar when you have different type of `JOIN`s)

```
SELECT {columns}
FROM {table_1}
INNER JOIN {table_2}
ON {table_1}.{common_key_1} = {table_2}.{common_key_2}
```

JOIN Statement - INNER

```
SELECT *
FROM country
JOIN city
ON country.country_id = city.country_id
WHERE country = 'United Kingdom';
```

JOIN Statement - INNER

- We can `JOIN ON` more than one table

```
SELECT *
FROM address
JOIN city
ON address.city_id = city.city_id
JOIN country
ON city.country_id = country.country_id
WHERE country = 'United Kingdom';
```

JOIN Statement - INNER

- We might want to get only the columns that belong to a certain table

```
SELECT address.*
FROM address
JOIN city
ON address.city_id = city.city_id
JOIN country
ON city.country_id = country.country_id
WHERE country = 'United Kingdom';
```

Aliasing

- Aliasing allows us to create temporary variables which we can reference in our query
  - Typically we alias our tables as just the first letter of the table name
  - We can actually omit the AS, but it's in this example for clarity

```
SELECT address, city
FROM address AS ad
JOIN city AS ci
ON ad.city_id = ci.city_id
JOIN country AS co
ON ci.country_id = co.country_id
WHERE country = 'United Kingdom';
```

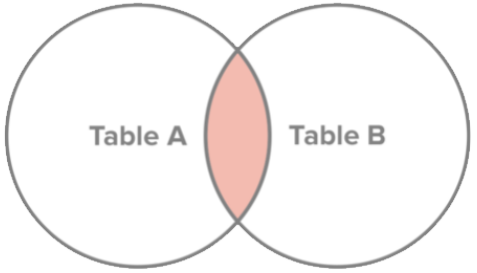
Practical - Part I

Go to the portal and complete the first practical in this lesson: “Using JOIN statements”

JOIN Statement - LEFT

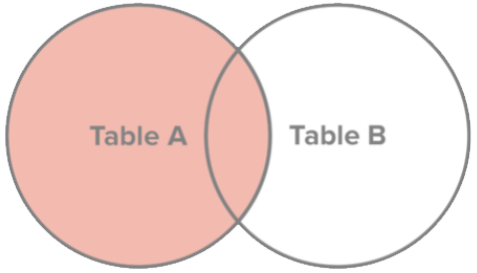
- There are four type of `JOIN`:

`INNER JOIN`



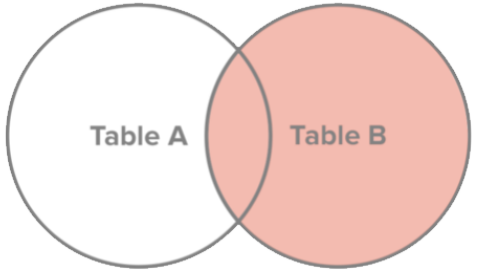
Select all records from Table A and Table B, where the join condition is met.

`LEFT JOIN`



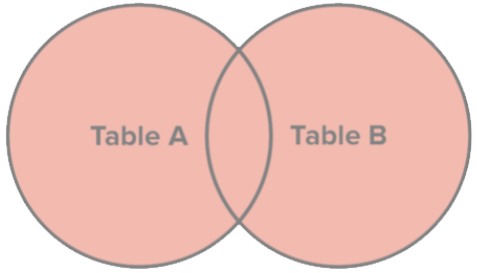
Select all records from Table A, along with records from Table B for which the join condition is met (if at all).

`RIGHT JOIN`



Select all records from Table B, along with records from Table A for which the join condition is met (if at all).

`FULL OUTER JOIN`



Select all records from Table A and Table B, regardless of whether the join condition is met or not.

JOIN Statement - LEFT

`LEFT JOIN`

1. The left table will have every row returned
2. Matches every row to the row in the right table (based on the `ON` condition)
    A. If the `ON` condition is True, columns from both tables are combined
    B. If the `ON` condition is False, a new row is still added but with a `NULL` value

JOIN Statement - LEFT

There might be some stores with no clients registered:

```
SELECT ad.address_id, cu.*
FROM address AS ad
LEFT JOIN customer AS cu
ON ad.address_id = cu.address_id
WHERE cu.customer_id IS NULL;
```

JOIN Statement - LEFT

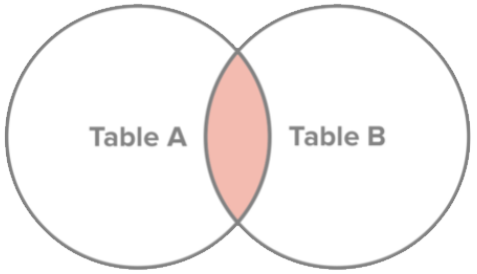
There might be some stores with no clients registered:

595	601	595	1	TERRENCE	GUNDERSON	TERRENCE.GUNDERSON@s...
596	602	596	1	ENRIQUE	FORSYTHE	ENRIQUE.FORSYTHE@sakil...
597	603	597	1	FREDDIE	DUGGAN	FREDDIE.DUGGAN@sakilac...
598	604	598	1	WADE	DELVALLE	WADE.DELVALLE@sakilacus...
599	605	599	2	AUSTIN	CINTRON	AUSTIN.CINTRON@sakilacu...
600	2	[null]	[null]	[null]	[null]	[null]
601	4	[null]	[null]	[null]	[null]	[null]
602	1	[null]	[null]	[null]	[null]	[null]
603	3	[null]	[null]	[null]	[null]	[null]

JOIN Statement - RIGHT

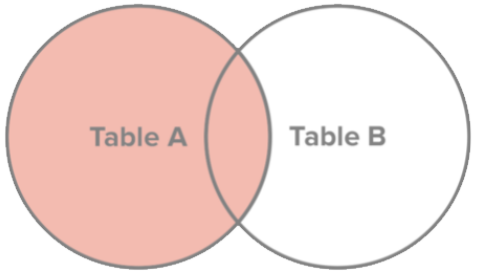
- There are four type of `JOIN`:

INNER JOIN



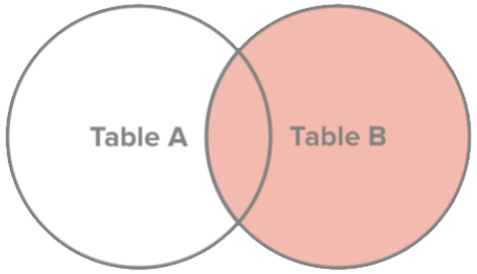
Select all records from Table A and Table B, where the join condition is met.

LEFT JOIN



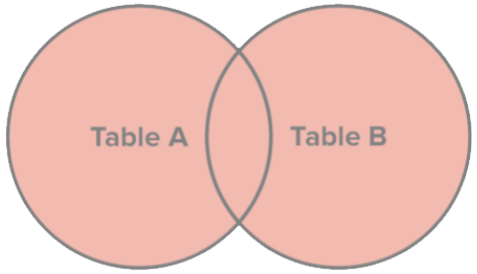
Select all records from Table A, along with records from Table B for which the join condition is met (if at all).

RIGHT JOIN



Select all records from Table B, along with records from Table A for which the join condition is met (if at all).

FULL OUTER JOIN



Select all records from Table A and Table B, regardless of whether the join condition is met or not.

JOIN Statement - RIGHT

`RIGHT JOIN`

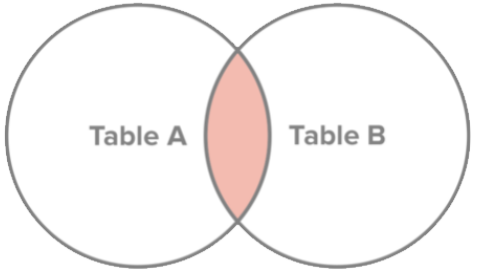
- Very similar to `LEFT JOIN`
- The output is similar, but just switched positions

```
SELECT as.address_id, cu.*
FROM customer AS cu
RIGHT JOIN address AS ad
ON cu.address_id = ad.address_id;
```

JOIN Statement - FULL OUTER

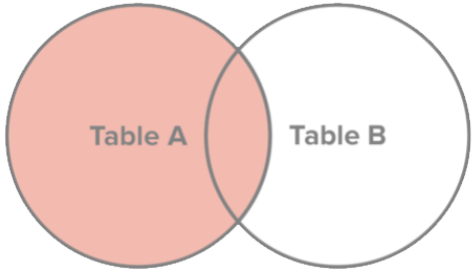
- There are four type of `JOIN`:

`INNER JOIN`



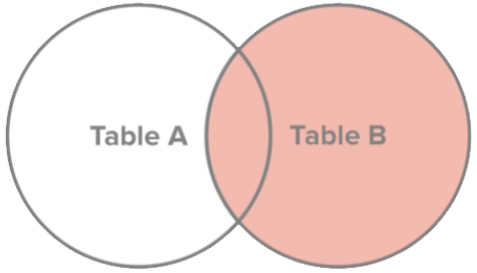
Select all records from Table A and Table B, where the join condition is met.

`LEFT JOIN`



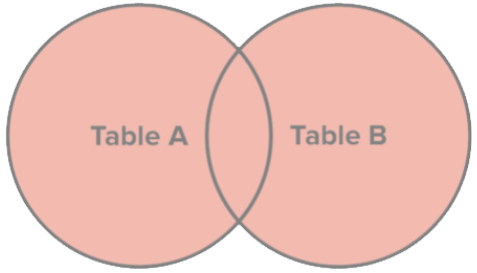
Select all records from Table A, along with records from Table B for which the join condition is met (if at all).

`RIGHT JOIN`



Select all records from Table B, along with records from Table A for which the join condition is met (if at all).

`FULL OUTER JOIN`



Select all records from Table A and Table B, regardless of whether the join condition is met or not.

JOIN Statement - FULL OUTER

```
FULL [OUTER] JOIN
```

- Take all data from both tables, regardless of the matches.
- Let's see the example from the previous slides (The employees exercise)

```
SELECT *
FROM employee_details AS det
FULL OUTER JOIN employee_salary AS sal
ON det.employee_id = sal.employee_id;
```

JOIN Statement - FULL OUTER

	employee_id integer	employee_name character varying (20)	employee_id integer	employee_name character varying (20)	salary integer
1	1	Mr. Pink	1	Mr. Pink	50000
2	2	Mr. Blonde	2	Mr. Blonde	48000
3	3	Mr. Orange	3	Mr. Orange	65000
4	4	Mr. White	[null]	[null]	[null]
5	5	Mr. Brown	[null]	[null]	[null]
6	6	Eddie	6	Eddie	90000
7	7	Joe	7	Joe	120000
8	[null]	[null]	8	Mr Blue	30000

JOIN Statement - FULL OUTER

```
FULL [OUTER] JOIN
```

- Rare use case
- Discussion about its uses: https://stackoverflow.com/questions/2094793/when-is-a-good-situation-to-use-a-full-outer-join

JOIN Statement

- There are four type of `JOIN`:

JOIN Statement

SELECT * FROM a
INNER JOIN b ON a.key = b.key

SELECT * FROM a
LEFT JOIN b ON a.key = b.key

SELECT * FROM a
RIGHT JOIN b ON a.key = b.key

SELECT * FROM a
LEFT JOIN b ON a.key = b.key
WHERE b.key IS NULL

SELECT * FROM a
RIGHT JOIN b ON a.key = b.key
WHERE a.key IS NULL

SELECT * FROM a
FULL JOIN b ON a.key = b.key

SELECT * FROM a
FULL JOIN b ON a.key = b.key
WHERE a.key IS NULL OR b.key IS NULL

https://www.postgresqltutorial.com/postgresql-joins/

Practical - Part II

Use the following tables to complete the Practical

customerID	customerName
1	Homer
2	Marge
3	Bart
4	Lisa
5	Maggie
6	Moe

orderID	customerID	item
1	1	Beer
2	2	Hair product
3	2	Dress
4	3	Juice
5	3	Magazine
6	6	Peanuts

Practical - Part II

1. Given the following SQL statement:

```
SELECT c.customerID, c.customerName, o.item
FROM customer AS c
INNER JOIN order AS o
ON c.customerID = o.customerID
```

What is the:

A. Number of columns in the returned table?
B. Number of rows in the returned table?
C. Number of times customerID “2” would show up?
D. Number of times customerID “5” would show up?

Practical - Part II

1. Given the following SQL statement:

```
SELECT c.customerID, c.customerName, o.item
FROM customer AS c
LEFT JOIN order AS o
ON c.customerID = o.customerID
```

What is the:

A. Number of columns in the returned table?
B. Number of rows in the returned table?
C. Number of times customerID “2” would show up?
D. Number of times customerID “5” would show up?