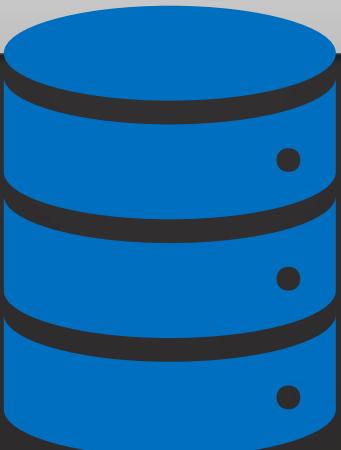


# Day 1



# Why learn SQL?



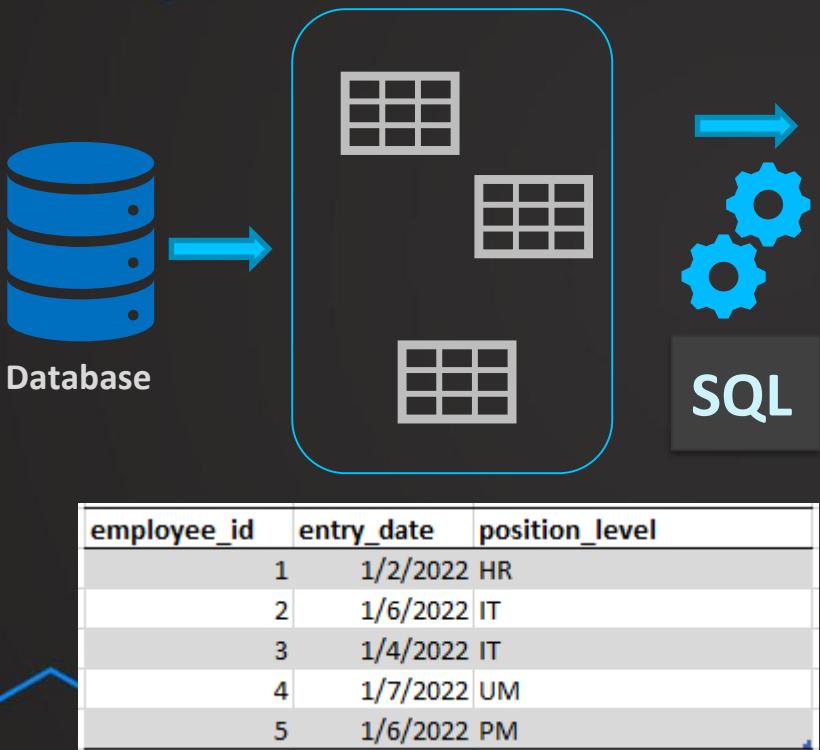
# What is SQL?

**SQL = Structured Query Language**

**Interact with databases**



# Using SQL



Departments

employee_id	entry_date	position_level
1	1/2/2022	HR
2	1/6/2022	IT
3	1/4/2022	IT
4	1/7/2022	UM
5	1/6/2022	PM

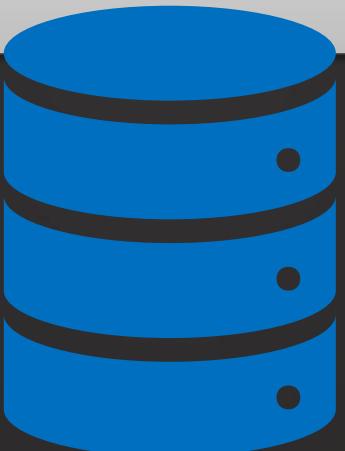
- Retrieve data
- Analyze data
- Define data
- Change data



# Why learn SQL?

- ✓ Data is everywhere and mostly in databases
- ✓ Maybe the most important skill as
  - Data Analyst,
  - Data Scientist,
  - Business Analyst
- ✓ Learning SQL is easy & intuitive
- ✓ Mastering SQL is a Career Booster

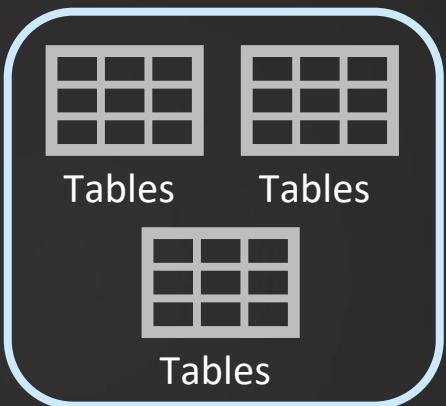
# What is a database?



# What is a database?



Database



```
SELECT  
FROM
```

*<column1>,  
<column2>, ...  
<table\_name>*

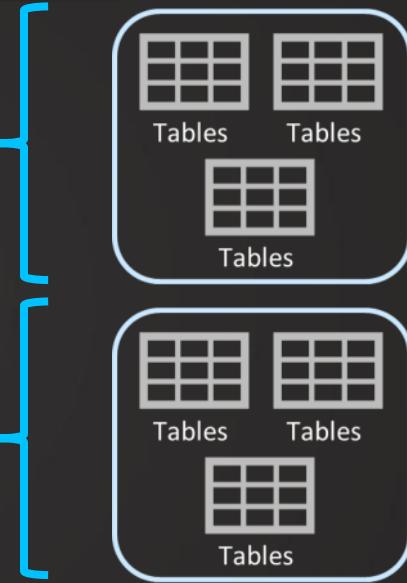
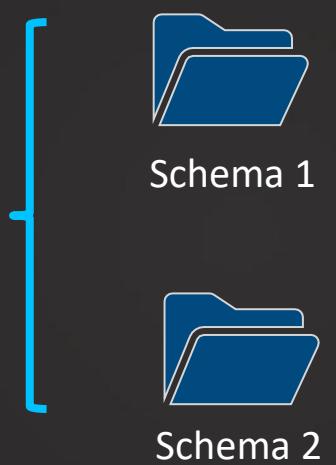
<i>id</i>	<i>date</i>	<i>product</i>	<i>customer_id</i>
1	1/2/2022	Fulltoss Tangy Tomato	2
2	1/2/2022	Chilli - Green, Organically Grown	2
3	1/2/2022	Masala Powder	5
4	1/2/2022	Cheese Cracker (Mcvities)	1
5	1/2/2022	Centre Filled Chocolate Cake	5

# What is a database?

Manged by



Database



Database management system (DBMS)



PostgreSQL

# What is a database?

## Database management system (DBMS)



## PostgreSQL

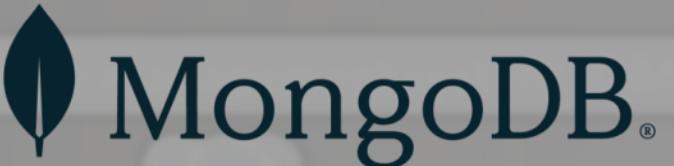
### Graphical Interface

A screenshot of the PgAdmin 4 graphical interface. The left pane shows a tree view of database objects under 'Servers' (PostgreSQL 9.5, PostgreSQL 13, PostgreSQL 14). The 'aircrafts' schema is selected. The right pane has tabs for 'Data Output', 'Explain', 'Messages', and 'Notifications'. A table titled 'aircraft\_code' is displayed with the following data:

	aircraft_code	model	range
1	773	Boeing 777-300	11100
2	763	Boeing 787-300	7900
3	SU9	Sukhoi Superjet-100	3000
4	320	Airbus A320-200	5700
5	321	Airbus A321-200	5600
6	319	Airbus A319-100	6700
7	733	Boeing 737-300	4200
8	CN1	Cessna 208 Caravan	1200
9	CR2	Bombardier CRJ-200	2700

## PgAdmin

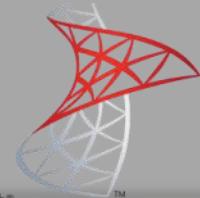
# Different DBMS



Postgre<sup>SQ</sup>L



MariaDB<sup>®</sup>



Microsoft<sup>®</sup>  
SQL Server<sup>®</sup>



PostgreSQL

# Different dialects

One language

All DBMS have slightly different dialects

The differences are small

PostgreSQL is the closest to the Standard SQL



PostgreSQ<sub>L</sub>

# Why PostgreSQL?

- ✓ PostgreSQL is the closest to the Standard SQL
- ✓ Most flexible & transition will be easiest
- ✓ Free to download & use
- ✓ Very popular
- ✓ The most advanced DBMS in the world

# Installation



Postgre<sup>SQ</sup>L

The screenshot shows the PgAdmin 4 interface. On the left, the 'Browser' panel displays a tree view of database objects under 'Servers (3)'. Under 'PostgreSQL 14', there are 'Databases (5)' including 'demo', 'DataWarehouseX', 'NedBikeShop', 'dem', and 'public'. Under 'Schemas (2)', there are 'bookings' and 'public'. Under 'Tables' (indicated by a yellow folder icon), there is a 'postgres' entry. On the right, the 'Query Editor' tab is active, showing the SQL command: `SELECT * FROM aircrafts;`. Below the query editor, the 'Data Output' tab is selected, displaying a table of aircraft data:

aircraft_code	model	range
773	Boeing 777-300	11100
763	Boeing 767-300	7900
8U9	Sukhoi Superjet 100	3000
920	Airbus A320-200	5700
921	Airbus A321-200	5600
319	Airbus A319-100	6700
733	Boeing 737-300	4200
CN1	Cessna 208 Caravan	1200
CR2	Bombardier CRJ-200	2700

PgAdmin

# The Project



YOU:  
Data Analyst

**GreenCYCLES**



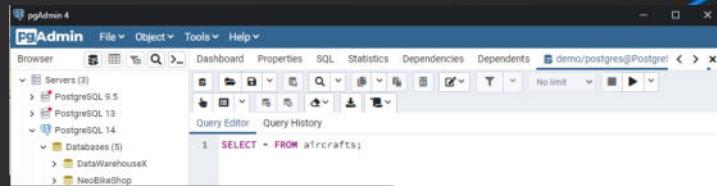
BUSINESS:  
Online Movie Rental Shop

# The Project



Data that is very  
important for the success!

YOU:  
Data Analyst

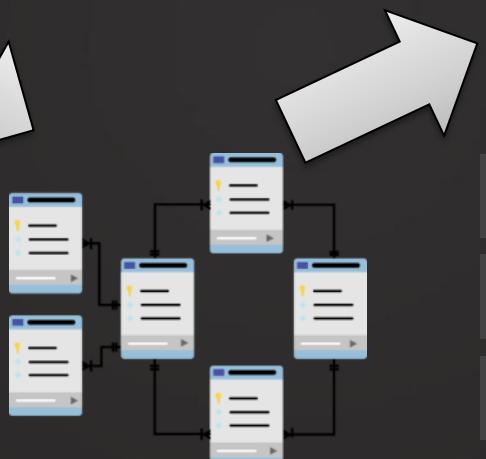


8	CNT	Cessna 208 Caravan	1200
9	CR2	Bombardier CRJ-200	2700

# Your job

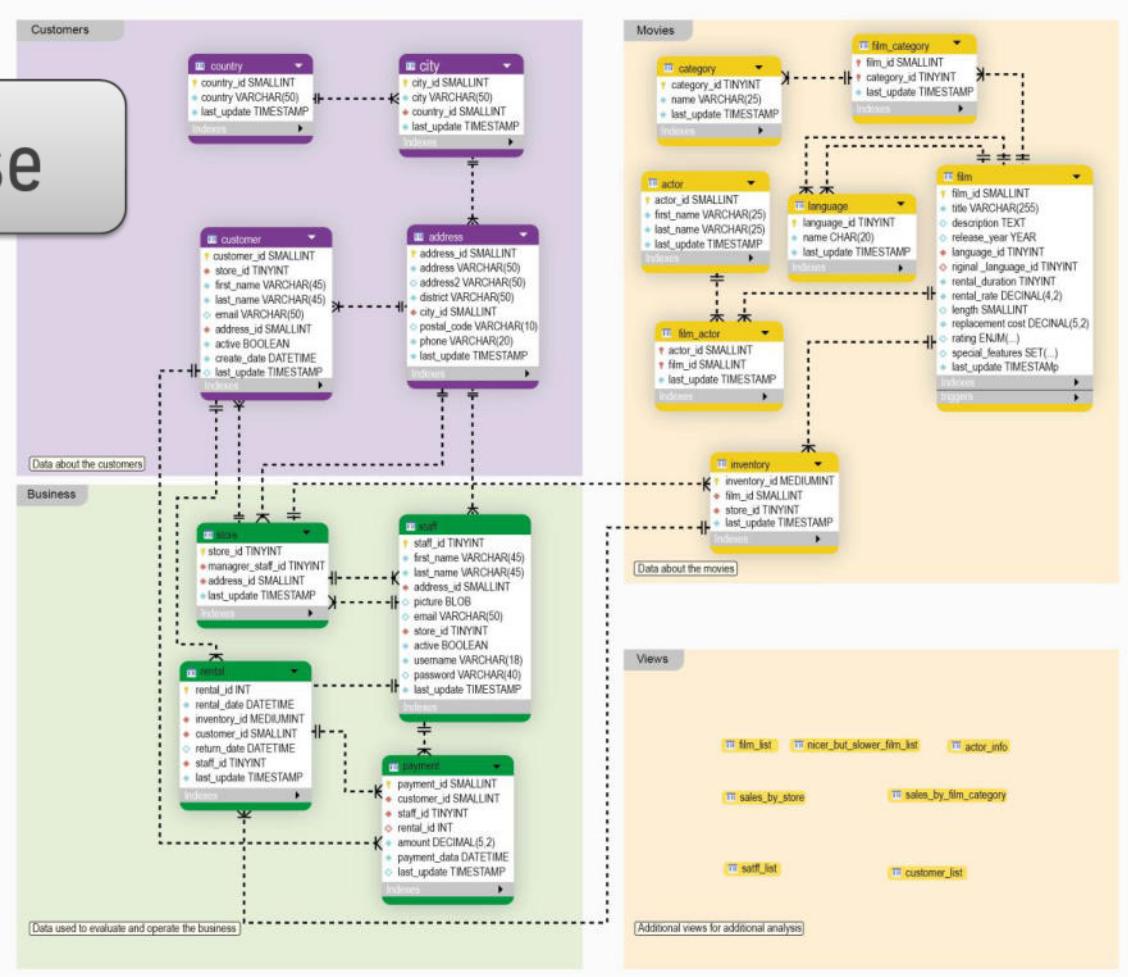


YOU:  
Data Analyst



- ✓ Help the company operate
- ✓ Gain insights
- ✓ Solve problems

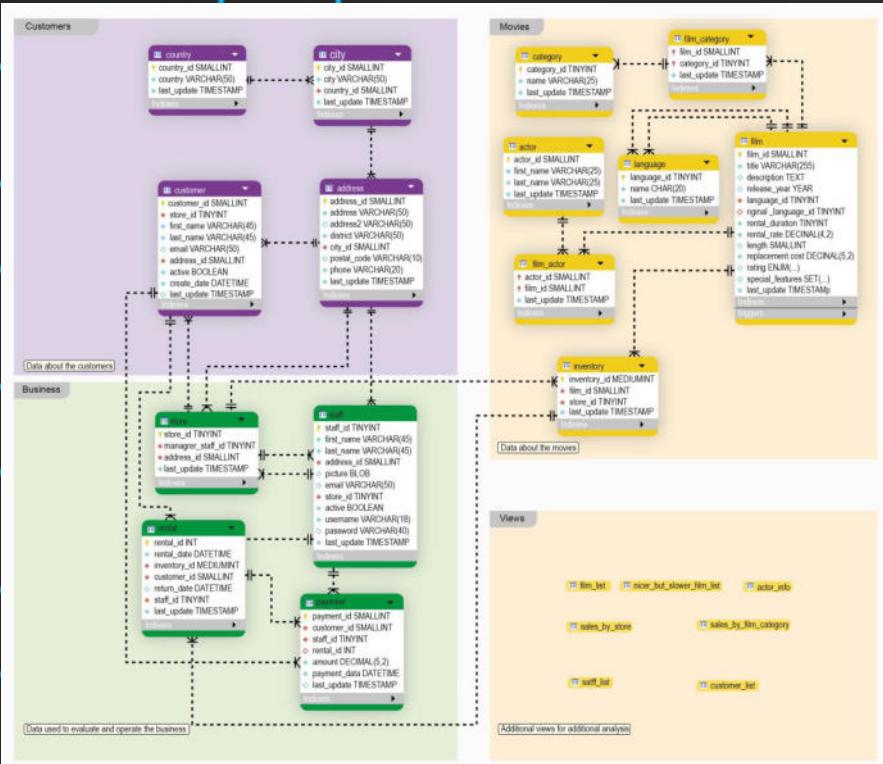
# The database



# Your challenges

- ✓ Explore meaningful data
- ✓ Get insights to make decisions
- ✓ Help to operate & navigate

You are responsible  
for the success!



# SELECT

- ✓ Most basic statement SQL
- ✓ Used to **select** and return data

# SYNTAX

```
SELECT  
column_name  
FROM table_name
```

# Example

```
SELECT  
first_name  
FROM actor
```

Data Output Explain Messages Notifications

	first_name	text	lock
1	PENELOPE		
2	NICK		
3	ED		
4	JENNIFER		
5	JOHNNY		

# Multiple columns

```
SELECT  
first_name,  
last_name  
FROM actor
```

Data Output Explain Messages Notifications

	first_name	last_name	
	text	text	
1	PENELOPE	GUINNESS	
2	NICK	WAHLBERG	
3	ED	CHASE	
4	JENNIFER	DAVIS	

# All columns

```
SELECT  
*  
FROM actor
```

Data Output Explain Messages Notifications

	actor_id [PK] integer	first_name text	last_name text	last_update timestamp with time zone	
1	1	PENELOPE	GUINNESS	2020-02-15 10:34:33+01	
2	2	NICK	WAHLBERG	2020-02-15 10:34:33+01	
3	3	ED	CHASE	2020-02-15 10:34:33+01	
4	4	JENNIFER	DAVIS	2020-02-15 10:34:33+01	

# Remarks!

## 1. Formatting doesn't matter!

```
SELECT  
first_name,  
last_name  
FROM actor
```

```
SELECT first_name, last_name  
FROM actor
```

```
SELECT first_name, last_name  
FROM actor
```

```
SELECT first_name, last_name FROM actor
```



Challenge

## Challenge

Your first day as a Data Analyst has started!

The Marketing Manager asks you for a **list** of all customers.

With first name, last name and the customer's **email address**.

**Write a SQL query to get that list!**

# ORDER BY

- ✓ Used to **order** results based on a column
- ✓ Alphabetically, numerically, chronologically etc.

# SYNTAX

```
SELECT  
column_name1,  
column_name2,  
FROM table_name  
ORDER BY column_name1
```

# Example

```
SELECT  
first_name,  
last_name  
FROM actor  
ORDER BY first_name
```

	first_name	last_name
1	ADAM	HOPPER
2	ADAM	GRANT
3	AL	GARLAND
4	ALAN	DREYFUSS

# DESC / ASC

```
SELECT  
first_name,  
last_name  
FROM actor  
ORDER BY first_name DESC
```

Data Output Explain Messages Notifications

	first_name text	last_name text	
1	ZERO	CAGE	
2	WOODY	HOFFMAN	
3	WOODY	JOLIE	

# DESC / ASC

```
SELECT  
first_name,  
last_name  
FROM actor  
ORDER BY first_name ASC
```

	first_name	last_name
1	ADAM	HOPPER
2	ADAM	GRANT
3	AL	GARLAND
4	ALAN	DREYFUSS

# Multiple columns

```
SELECT  
first_name,  
last_name  
FROM actor  
ORDER BY first_name, last_name
```

Data Output Explain Messages Notifications

	first_name	last_name
1	ADAM	GRANT
2	ADAM	HOPPER
3	AL	GARLAND

# DESC / ASC

```
SELECT  
first_name,  
last_name  
FROM actor  
ORDER BY first_name DESC, last_name
```

	first_name	last_name
1	ZERO	CAGE
2	WOODY	HOFFMAN
3	WOODY	JOLIE



Challenge

# SELECT DISTINCT

- ✓ Used to SELECT the DISTINCT values in a table

# SYNTAX

```
SELECT DISTINCT  
column_name1  
FROM table_name
```

# Example

```
SELECT DISTINCT  
first_name  
FROM actor
```

Data Output Explain

	first_name	text	lock
1	SIDNEY		
2	WOODY		
3	GOLDIE		
4	CHRIS		

# Example

```
SELECT DISTINCT  
first_name  
FROM actor  
ORDER BY first_name
```

	first_name	
1	ADAM	
2	AL	
3	ALAN	
4	ALBERT	

# Multiple columns

```
SELECT DISTINCT  
first_name,  
last_name  
FROM actor  
ORDER BY first_name
```

Note!

Distinct in terms of  
all selected columns!

	first_name text	last_name text
1	ADAM	GRANT
2	ADAM	HOPPER
3	AL	GARLAND
4	ALAN	DREYFUSS

# Multiple columns

```
SELECT  
first_name,  
last_name  
FROM actor  
ORDER BY first_name, last_name
```

Data Output Explain Messages Notifications

	first_name	last_name
1	ADAM	GRANT
2	ADAM	HOPPER
3	AL	GARLAND



Challenge

# Challenge

A marketing team member asks you about the different prices that have been paid.

To make it easier for them order the prices from high to low.

**Write a SQL query to get the different prices!**

## Result

Data Output		Explain
	amount	
	numeric (5,2)	lock
1	11.99	
2	10.99	
3	9.99	
4	9.98	
5	8.99	
6	8.97	
7	7.99	
8	7.98	

# LIMIT

- ✓ Used to **LIMIT** the number of rows in the output
- ✓ Always at the very end of your query
- ✓ Can help to get a quick idea about a table

# SYNTAX

```
SELECT  
column_name1,  
column_name2  
FROM table_name  
LIMIT n
```

# SYNTAX

```
SELECT  
first_name  
FROM actor  
LIMIT 4
```

Data Output		Explain
	first_name	text
1	PENELOPE	
2	NICK	
3	ED	
4	JENNIFER	

# Example

```
SELECT  
first_name  
FROM actor  
ORDER BY first_name  
LIMIT 4
```

Data Output		Explain
	first_name	
1	ADAM	
2	ADAM	
3	AL	
4	ALAN	

# Multiple columns

```
SELECT DISTINCT  
first_name,  
last_name  
FROM actor  
ORDER BY first_name
```

Note!

Distinct in terms of  
all selected columns!

	first_name text	last_name text
1	ADAM	GRANT
2	ADAM	HOPPER
3	AL	GARLAND
4	ALAN	DREYFUSS



Challenge

# Challenge

A marketing team member asks you about the different prices that have been paid.

To make it easier for them order the prices from high to low.

**Write a SQL query to get the different prices!**

## Result

Data Output		Explain
	amount	
	numeric (5,2)	lock
1	11.99	
2	10.99	
3	9.99	
4	9.98	
5	8.99	
6	8.97	
7	7.99	
8	7.98	

# COUNT

- ✓ Used to COUNT the number of rows in a output
- ✓ Used often in combination with grouping & filtering

# SYNTAX

```
SELECT  
COUNT(*)  
FROM table_name
```

# SYNTAX

```
SELECT  
COUNT(column_name)  
FROM table_name
```

Note!

Nulls will not be  
counted in that case!

# Example

```
SELECT  
COUNT(first_name)  
FROM actor
```

Data Output Explain

	count	bigint
1	200	

# Example

```
SELECT  
COUNT(*)  
FROM actor
```

Data Output    Explain

	count	bigint
1	200	

# Example

```
SELECT  
COUNT(DISTINCT first_name)  
FROM actor
```

Data Output	
	count
	bigint
1	128

# Multiple columns

```
SELECT DISTINCT  
first_name,  
last_name  
FROM actor  
ORDER BY first_name
```

Note!

Distinct in terms of  
all selected columns!

	first_name text	last_name text
1	ADAM	GRANT
2	ADAM	HOPPER
3	AL	GARLAND
4	ALAN	DREYFUSS



Challenge  
for today

## Challenge for today

1. Create a list of all the **distinct districts** customers are from.
2. What is the **latest rental date**?
3. How many films does the company have?
4. How many **distinct last names** of the customers are there?

## Results

### Result 1

Data Output	Explain	Me
district	text	lock
1	Aden	
2	Eastern Visayas	
3	Vaduz	
4	Tokat	
5	Anzotegui	
6	Saint Denis	
7	Chittanam	
8	Chihuahua	

### Result 2

Data Output	Explain	Me
rental_date	timestamp with time zone	lock

1 2020-02-14 16:16:03+01

### Result 3

Data Output	Explain	Me
count	bigint	lock

1 1000

### Result 4

Data Output	Explain	Me
count	bigint	lock

1 599

## Solution 1

1. Create a list of all the distinct districts customers are from.

```
SELECT DISTINCT
    district
    FROM address
```

## Solution 2

2. What is the **latest rental date**?

```
SELECT
    rental_date
    FROM rental
    ORDER BY rental_date DESC
    LIMIT 1
```

## Challenge for today

1. Create a list of all the **distinct districts** customers are from.
2. What is the **latest rental date**?
3. **How many films** does the company have?
4. **How many distinct last names** of the customers are there?

# Results

Result 1

	Data Output	Explain	Me
	district text		🔒
1	Aden		
2	Eastern Visayas		
3	Vaduz		
4	Tokat		
5	Anzotegui		
6	Saint-Denis		
7	Chollanam		
8	Chihuahua		

Result 2

	Data Output	Explain	Me
	rental_date timestamp with time zone		🔒
1	2020-02-14 16:16:03+01		

Result 3

	Data Output
	count bigint
1	1000

Result 4

	Data Output
	count bigint
1	599

# Solution 1

1. Create a list of all the **distinct districts** customers are from.

```
SELECT DISTINCT  
district  
FROM address
```

# Solution 2

2. What is the [latest rental date](#)?

```
SELECT  
    rental_date  
FROM    rental  
ORDER  BY rental_date DESC  
LIMIT  1
```

## Solution 2

3. How many films does the company have?

```
SELECT  
COUNT(*)  
FROM film
```

## Solution 2

4. How many distinct last names of the customers are there?

```
SELECT  
COUNT(DISTINCT last_name)  
FROM customer
```

# Challenge

A marketing team member asks you about the different prices that have been paid.

To make it easier for them order the prices from high to low.

**Write a SQL query to get the different prices!**

## Result

Data Output		Explain
	amount	
	numeric (5,2)	lock
1	11.99	
2	10.99	
3	9.99	
4	9.98	
5	8.99	
6	8.97	
7	7.99	
8	7.98	

## Challenge

You need to help the Marketing team to work more easily.

The Marketing Manager asks you to order the customer list by the **last name**.

They want to start from "Z" and work towards "A".

In case of the same last name the order should be based on the first name – also from "Z" to "A".

**Write a SQL query to get that list!**

# Day 2



# WHERE

✓ Used to **FILTER** the data in the output

✓ Always after **FROM**

# SYNTAX

```
SELECT  
column_name1,  
column_name2  
FROM table_name  
WHERE condition
```

# SYNTAX

```
SELECT  
    *  
FROM payment  
WHERE amount = 10.99
```

Data Output							Explain	Messages	Notifications
	payment_id	customer_id	staff_id	rental_id	amount	payment_date			
1	16073	276	1	860	10.99	2020-01-30 02:13:42.996577+01			
2	16125	304	1	135	10.99	2020-01-25 21:27:24.996577+01			
3	16358	433	1	691	10.99	2020-01-29 00:29:52.996577+01			
4	16389	191	1	169	10.99	2020-01-26 01:51:10.996577+01			

# SYNTAX

```
SELECT  
first_name,  
last_name  
FROM customer  
WHERE first_name = 'ADAM'
```

Data Output Explain Messages Notifications

	first_name	last_name	
	text	text	
1	ADAM	GOOCH	

# SYNTAX

```
SELECT  
first_name,  
last_name  
FROM customer  
WHERE first_name = 'ADAM'
```

Data Output Explain Messages Notifications

	first_name	last_name	
	text	text	
1	ADAM	GOOCH	



Challenge

# Challenge

How many payment were made by the customer with customer\_id = 100?

What is the last name of our customer with first name 'ERICa'?

**Write a SQL query to get the answers**

Result

Data Output		Explain
	count	bigint
1	24	

Data Output		Explain	Message
	first_name	last_name	
1	ERICa	MATTHEWS	

# SYNTAX

```
SELECT  
    *  
FROM payment  
WHERE amount > 10.99
```

Data Output							Explain	Messages	Notifications
	payment_id	customer_id	staff_id	rental_id	amount	payment_date			
1	17055	196	2	106	11.99	2020-01-25 17:46:45.996577+01			
2	17354	305	1	2166	11.99	2020-02-17 23:19:47.996577+01			
3	20403	362	1	14759	11.99	2020-03-21 22:57:24.996577+01			

# SYNTAX

```
SELECT  
*  
FROM payment  
WHERE amount < 10.99
```

Data Output Explain Messages Notifications

	payment_id integer	customer_id smallint	staff_id smallint	rental_id integer	amount numeric (5,2)	payment_date timestamp with time zone	lock
1	16050	269	2	7	1.99	2020-01-24 22:40:19.996577+01	
2	16051	269	1	98	0.99	2020-01-25 16:16:50.996577+01	
3	16052	269	2	678	6.99	2020-01-28 22:44:14.996577+01	

# SYNTAX

```
SELECT  
*  
FROM payment  
WHERE amount < 10.99  
ORDER BY amount DESC
```

Data Output Explain Messages Notifications

	payment_id integer	customer_id smallint	staff_id smallint	rental_id integer	amount numeric (5,2)	payment_date timestamp with time zone
1	22124	542	2	14982	9.99	2020-03-22 06:49:21.996577+01
2	22127	543	2	11241	9.99	2020-03-02 12:57:50.996577+01
3	29567	51	1	3525	9.99	2020-04-06 00:31:05.996577+02

# SYNTAX

```
SELECT  
*  
FROM payment  
WHERE amount <= 10.99  
ORDER BY amount DESC
```

Data Output Explain Messages Notifications

	payment_id integer	customer_id smallint	staff_id smallint	rental_id integer	amount numeric (5,2)	payment_date timestamp with time zone	lock
1	23952	136	2	11992	10.99	2020-03-17 17:55:48.996577+01	
2	31310	212	2	4708	10.99	2020-04-08 11:27:45.996577+02	
3	20817	404	2	15290	10.99	2020-03-22 18:56:28.996577+01	

# SYNTAX

```
SELECT  
*  
FROM payment  
WHERE amount >= 10.99  
ORDER BY amount DESC
```

Data Output Explain Messages Notifications

	payment_id integer	customer_id smallint	staff_id smallint	rental_id integer	amount numeric (5,2)	payment_date timestamp with time zone	lock
1	20403	362	1	14759	11.99	2020-03-21 22:57:24.9996577+01	
2	24866	237	2	11479	11.99	2020-03-02 21:46:39.9996577+01	
3	17055	196	2	106	11.99	2020-01-25 17:46:45.9996577+01	

# SYNTAX

```
SELECT  
*  
FROM payment  
WHERE amount != 10.99
```

Data Output							Explain	Messages	Notifications
	payment_id	customer_id	staff_id	rental_id	amount	payment_date			
1	16050	269	2	7	1.99	2020-01-24 22:40:19.996577+01			
2	16051	269	1	98	0.99	2020-01-25 16:16:50.996577+01			
3	16052	269	2	678	6.99	2020-01-28 22:44:14.996577+01			

# SYNTAX

```
SELECT  
*  
FROM payment  
WHERE amount <> 10.99
```

Data Output							Explain	Messages	Notifications
	payment_id	customer_id	staff_id	rental_id	amount	payment_date			
1	16050	269	2	7	1.99	2020-01-24 22:40:19.996577+01			
2	16051	269	1	98	0.99	2020-01-25 16:16:50.996577+01			
3	16052	269	2	678	6.99	2020-01-28 22:44:14.996577+01			

# SYNTAX

```
SELECT  
*  
FROM payment  
WHERE amount <= 10.99  
ORDER BY amount DESC
```

# SYNTAX

```
SELECT  
first_name,  
last_name  
FROM customer  
WHERE first_name is null
```

Side note!

Case sensitive or not?

# SYNTAX

```
SELECT  
first_name,  
last_name  
FROM customer  
WHERE first_name is not null
```

Side note!

Case sensitive or not?

# SYNTAX

```
select  
first_name,  
last_name  
FROM customer  
WHERE first_name = 'ADAM'
```

Data Output Explain Messages Notifications

	first_name	last_name	
	text	text	
1	ADAM	GOOCH	

Side note!

Case sensitive or not?

# SYNTAX

```
seLEcT  
first_name,  
last_name  
FROM customer  
WHERE first_name = 'ADAM'
```

Data Output Explain Messages Notifications

	first_name	last_name	
	text	text	
1	ADAM	GOOCH	

Side note!

Case sensitive or not?

# SYNTAX

```
select  
FIRst_nAme,  
last_name  
FROM customer  
WHERE first_name = 'ADAM'
```

Data Output Explain Messages Notifications

	first_name	last_name	
	text	text	
1	ADAM	GOOCH	

Side note!

Case sensitive or not?

# SYNTAX

```
select  
  "first_name",  
  last_name  
FROM customer  
WHERE first_name = 'ADAM'
```

Data Output Explain Messages Notifications

	first_name	last_name	
	text	text	
1	ADAM	GOOCH	

# SYNTAX

```
select  
  "First_name",  
    last_name  
 FROM customer  
 WHERE first_name = 'ADAM'
```

Data Output Explain Messages Notifications

ERROR: column "First\_name" does not exist  
LINE 3: "First\_name",  
 ^

HINT: Perhaps you meant to reference the column "customer.first\_name".  
SQL state: 42703  
Character: 10



Challenge

# Challenge

The inventory manager asks you how rentals have not been returned yet (return\_date is null).

The sales manager asks you how for a list of all the payment\_ids with an amount less than or equal to \$2. Include payment\_id and the amount.

**Write a SQL query to get the answers!**

Result

Data Output		Explain
	count	bigint
1	183	

Data Output			Explain	Message
	payment_id	amount	numeric (5,2)	
1	16050		1.99	
2	16051		0.99	
3	16053		0.99	
4	16056		1.99	
5	16059		0.99	

# AND / OR

- ✓ Used to connect **two conditions**

# SYNTAX

```
SELECT  
column_name1,  
column_name2  
FROM table_name  
WHERE condition1  
AND condition2  
AND condition3
```

Note!

ALL conditions must be  
true!

# SYNTAX

```
SELECT  
*  
FROM payment  
WHERE amount = 10.99  
AND customer_id = 426
```

Data Output							Explain	Messages	Notifications
	payment_id	customer_id	staff_id	rental_id	amount	payment_date			
1	26978	426	1	7527	10.99	2020-04-27 20:42:54.996577+02			
2	26983	426	2	10172	10.99	2020-04-30 22:58:17.996577+02			

# SYNTAX

```
SELECT  
*  
FROM payment  
WHERE amount = 10.99  
OR amount = 9.99
```

Data Output Explain Messages Notifications

	payment_id	customer_id	staff_id	rental_id	amount	payment_date
	integer	smallint	smallint	integer	numeric (5,2)	timestamp with time zone
1	16073	276	1	860	10.99	2020-01-30 02:13:42.996577+01
2	16125	304	1	135	10.99	2020-01-25 21:27:24.996577+01
3	16160	318	1	224	9.99	2020-01-26 09:46:53.996577+01

# SYNTAX

```
SELECT  
*  
FROM payment  
WHERE amount = 10.99 OR 9.99
```

Data Output Explain Messages Notifications

ERROR: argument of OR must be type boolean, not type numeric

LINE 6: OR 9.99

^

SQL state: 42804

Character: 53

# SYNTAX

```
SELECT  
*  
FROM payment  
WHERE amount = 10.99  
OR amount = 9.99  
AND customer_id = 426
```

Data Output Explain Messages Notifications

	payment_id integer	customer_id smallint	staff_id smallint	rental_id integer	amount numeric (5,2)	payment_date timestamp with time zone	lock
1	16073	276	1	860	10.99	2020-01-30 02:13:42.996577+01	lock
2	16125	304	1	135	10.99	2020-01-25 21:27:24.996577+01	lock
3	16358	433	1	691	10.99	2020-01-29 00:29:52.996577+01	lock

# SYNTAX

```
SELECT  
*  
FROM payment  
WHERE amount = 10.99  
OR (amount = 9.99  
AND customer_id = 426)
```

Data Output Explain Messages Notifications

	payment_id integer	customer_id smallint	staff_id smallint	rental_id integer	amount numeric (5,2)	payment_date timestamp with time zone	
1	16073	276	1	860	10.99	2020-01-30 02:13:42.996577+01	
2	16125	304	1	135	10.99	2020-01-25 21:27:24.996577+01	
3	16358	433	1	691	10.99	2020-01-29 00:29:52.996577+01	

# SYNTAX

```
SELECT  
*  
FROM payment  
WHERE (amount = 10.99  
OR amount = 9.99)  
AND customer_id = 426
```

Data Output Explain Messages Notifications

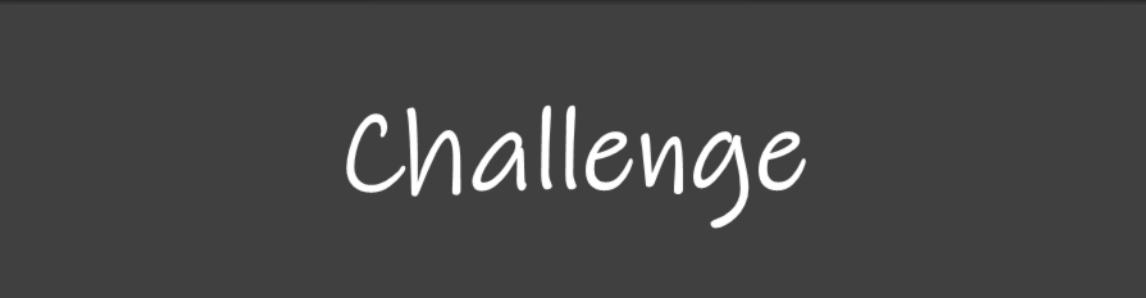
	payment_id integer	customer_id smallint	staff_id smallint	rental_id integer	amount numeric (5,2)	payment_date timestamp with time zone	lock
1	26978	426	1	7527	10.99	2020-04-27 20:42:54.996577+02	
2	26983	426	2	10172	10.99	2020-04-30 22:58:17.996577+02	

# SYNTAX

```
SELECT  
first_name,  
last_name  
FROM customer  
WHERE first_name = 'ADAM'
```

Data Output Explain Messages Notifications

	first_name	last_name	
	text	text	
1	ADAM	GOOCH	



Challenge

# Challenge

The suppcity manager asks you about a list of all the payment of the customer 322, 346 and 354 where the amount is either less than \$2 or greater than \$10.

It should be ordered by the customer first (ascending) and then as second condition order by amount in a descending order.

**Write a SQL query to get the answers!**

Result

Data Output							Explain	Messages	Notifications
	payment_id	customer_id	staff_id	rental_id	amount	payment_date			
	integer	smallint	smallint	integer	numeric (5,2)	timestamp with time zone			
1	25784	322	2	3627	1.99	2020-04-06 05:47:51.996577+02			
2	25794	322	1	9252	1.99	2020-04-30 13:48:25.996577+02			
3	16167	322	2	166	0.99	2020-01-26 02:17:37.996577+01			

# BETWEEN ... AND ...

- ✓ Used to filter a range of values

# SYNTAX

```
SELECT  
payment_id,  
amount  
FROM payment  
WHERE amount NOT BETWEEN 1.99 AND 6.99
```

Data Output Explain Messages

	payment_id	amount
	integer	numeric (5,2)
1	16051	0.99
2	16053	0.99
3	16058	8.99
4	16059	0.99

# SYNTAX

```
SELECT  
payment_id,  
amount  
FROM payment  
WHERE amount BETWEEN '2020-01-24' AND '2020-01-26'  
      'YYYY-MM-DD'
```

Data Output Explain Messages Notifications

	payment_id	payment_date
	integer	timestamp with time zone
1	16050	2020-01-24 22:40:19.996577+01
2	16051	2020-01-25 16:16:50.996577+01
3	16059	2020-01-25 03:47:17.996577+01
4	16063	2020-01-25 19:14:47.996577+01

# SYNTAX

```
SELECT  
payment_id,  
amount  
FROM payment  
WHERE amount BETWEEN '2020-01-24 0:00' AND '2020-01-26 0:00'
```

Data Output Explain Messages Notifications

	payment_id	payment_date
	integer	timestamp with time zone
1	16050	2020-01-24 22:40:19.996577+01
2	16051	2020-01-25 16:16:50.996577+01
3	16059	2020-01-25 03:47:17.996577+01
4	16063	2020-01-25 19:14:47.996577+01

# SYNTAX

```
SELECT  
payment_id,  
amount  
FROM payment  
WHERE amount BETWEEN '2020-01-24 0:00' AND '2020-01-26 12:00'
```

Data Output Explain Messages Notifications

	payment_id	payment_date
	integer	timestamp with time zone
1	16050	2020-01-24 22:40:19.996577+01
2	16051	2020-01-25 16:16:50.996577+01
3	16059	2020-01-25 03:47:17.996577+01
4	16063	2020-01-25 19:14:47.996577+01



Challenge

## Challenge

There have been 6 complaints of customers about their payments.

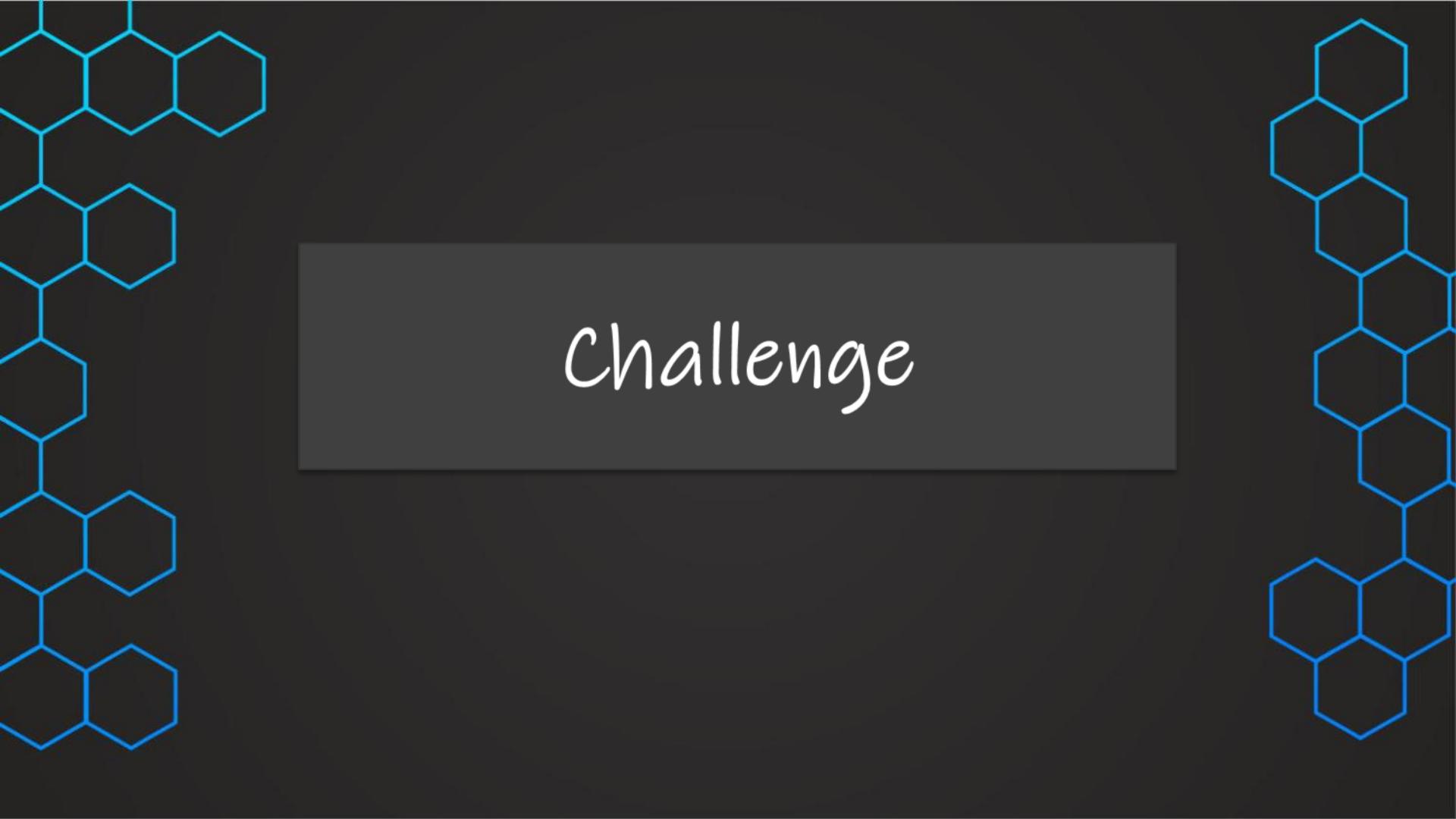
customer\_id: 12,25,67,93,124,234

The concerned payments are all the payments of these customers with amounts 4.99, 7.99 and 9.99 in January 2020.

**Write a SQL query to get a list of the concerned payments!**

Result

**It should be 7 payments!**



Challenge

## Challenge

There have been some faulty payments and you need to help to find out how many payments have been affected.

How many payments have been made on January 26th and 27th 2020 (including entire 27th) with an amount between 1.99 and 3.99?

**Write a SQL query to get the answers!**

Result

Data Output		Explain	More
	count	bigint	
1	45		

# LIKE

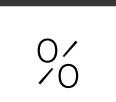
✓ Used to filter by matching against a pattern

✓ Use wildcards:



any single character

✓ Use wildcards:



any sequence of characters

# SYNTAX

```
SELECT  
*  
FROM actor  
WHERE first_name LIKE 'A%'
```

Data Output Explain Messages Notifications

	actor_id [PK] integer	first_name text	last_name text	last_update timestamp with time zone
1	29	ALEC	WAYNE	2020-02-15 10:34:33+01
2	34	AUDREY	OLIVIER	2020-02-15 10:34:33+01
3	49	ANNE	CRONYN	2020-02-15 10:34:33+01

Note!

It's case-sensitive

# SYNTAX

```
SELECT  
first_name  
FROM actor  
WHERE first_name LIKE 'a%'
```

Data Output				Explain	Messages	Notifications
actor_id	first_name	last_name	last_update			

Note!

It's case-sensitive

# SYNTAX

```
SELECT  
first_name  
FROM actor  
WHERE first_name ILIKE 'a%'
```

Data Output Explain Messages Notifications

	actor_id [PK] integer	first_name text	last_name text	last_update timestamp with time zone
1	29	ALEC	WAYNE	2020-02-15 10:34:33+01
2	34	AUDREY	OLIVIER	2020-02-15 10:34:33+01
3	49	ANNE	CRONYN	2020-02-15 10:34:33+01

Note!

ILIKE is case-INsensitive

# SYNTAX

```
SELECT  
*  
FROM actor  
WHERE first_name LIKE 'A%'
```

Data Output Explain Messages Notifications

	actor_id [PK] integer	first_name text	last_name text	last_update timestamp with time zone
1	29	ALEC	WAYNE	2020-02-15 10:34:33+01
2	34	AUDREY	OLIVIER	2020-02-15 10:34:33+01
3	49	ANNE	CRONYN	2020-02-15 10:34:33+01

Note!

It's case-sensitive

# SYNTAX

```
SELECT  
*  
FROM actor  
WHERE first_name LIKE '%A%'
```

Data Output Explain Messages Notifications

	actor_id [PK] integer	first_name text	last_name text	last_update timestamp with time zone
1	7	GRACE	MOSTEL	2020-02-15 10:34:33+01
2	8	MATTHEW	JOHANSSON	2020-02-15 10:34:33+01
3	10	CHRISTIAN	GABLE	2020-02-15 10:34:33+01

# SYNTAX

```
SELECT  
*  
FROM actor  
WHERE first_name LIKE '_A%'
```

Data Output					Explain	Messages	Notifications
	actor_id [PK] integer	first_name text	last_name text	last_update timestamp with time zone			
1	8	MATTHEW	JOHANSSON	2020-02-15 10:34:33+01			
2	12	KARL	BERRY	2020-02-15 10:34:33+01			
3	18	DAN	TORN	2020-02-15 10:34:33+01			
4	23	SANDRA	KILMER	2020-02-15 10:34:33+01			

# SYNTAX

SELECT

\*

FROM actor

WHERE first\_name LIKE '\_ \_A%'

Data Output Explain Messages Notifications

	actor_id [PK] integer	first_name text	last_name text	last_update timestamp with time zone
1	7	GRACE	MOSTEL	2020-02-15 10:34:33+01
2	13	UMA	WOOD	2020-02-15 10:34:33+01
3	48	FRANCES	DAY-LEWIS	2020-02-15 10:34:33+01

# SYNTAX

```
SELECT  
*  
FROM actor  
WHERE first_name NOT LIKE '%A%'
```

Data Output						Explain	Messages	Notifications
	actor_id [PK] integer	first_name text		last_name text		last_update timestamp with time zone		
1	1	PENELOPE		GUINNESS		2020-02-15 10:34:33+01		
2	2	NICK		WAHLBERG		2020-02-15 10:34:33+01		
3	3	ED		CHASE		2020-02-15 10:34:33+01		

# SYNTAX

```
SELECT  
*  
FROM payment  
WHERE amount = 10.99  
AND customer_id = 426
```

Data Output Explain Messages Notifications

	payment_id integer	customer_id smallint	staff_id smallint	rental_id integer	amount numeric (5,2)	payment_date timestamp with time zone	lock
1	26978	426	1	7527	10.99	2020-04-27 20:42:54.996577+02	lock
2	26983	426	2	10172	10.99	2020-04-30 22:58:17.996577+02	lock



Challenge

# Challenge

You need to help the inventory manager to find out:

How many movies are there that contain the "Documentary" in the description?

**Write a SQL query to get the answers!**

Result

Data Output	
	count
	bigint
1	101

# Challenge

How many customers are there with a first name that is 3 letters long and either an 'X' or a 'Y' as the last letter in the last name?

**Write a SQL query to get the answers!**

Result

Data Output Expl

count	bigint
1	3

# Commenting & Aliases

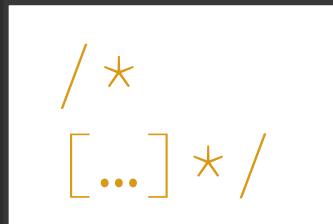
✓ Comment to make code more readable & understandable

✓ Use



Single line comment

✓ Use



/\*  
[...]\*/

A white box containing the multiple-line comment symbol, represented by a forward slash (\*), followed by an asterisk (\*), and then three dots [...] enclosed in brackets, followed by another asterisk (\*).

Multiple lines comment

Note!

Use comments to  
explain your code!

# SYNTAX

```
-- 2020/07/01 by Nikolai  
-- Query that filters amount
```

```
SELECT  
*  
FROM payment  
WHERE amount = 10.99
```

Note!

Comments will not be  
processed as code!

# SYNTAX

```
/* 2020/07/01 by Nikolai  
Query to filter by amount*/
```

```
SELECT  
*  
FROM payment  
WHERE amount = 10.99
```

# SYNTAX

```
SELECT  
*  
FROM payment  
WHERE amount = 10.99  
--AND customer_id = 426
```

# SYNTAX

```
SELECT  
*  
FROM payment  
--WHERE amount = 10.99  
--AND customer_id = 426
```

# SYNTAX

```
SELECT  
*  
FROM payment  
/*WHERE amount = 10.99  
AND customer_id = 426*/
```

# SYNTAX

```
SELECT  
payment_id AS invoice_no  
FROM payment
```

Data Output Explain Messages

	invoice_no	amount
	integer	numeric (5,2)
1	16050	1.99
2	16051	0.99
3	16052	6.99

# SYNTAX

```
SELECT  
COUNT(*)  
FROM payment  
WHERE amount = 10.99  
AND customer_id = 426
```



Challenge  
for today

## Challenge for today

1. How many movies are there that contain 'Saga' in the description and where the title starts either with 'A' or ends with 'R'?  
Use the alias 'no\_of\_movies'.
2. Create a list of all customers where the first name contains 'ER' and has an 'A' as the second letter.  
Order the results by the last name descendingly.
3. How many payments are there where the amount is either 0 or is between 3.99 and 7.99 and in the same time has happened on 2020-05-01.

# Results

Result 1

Data Output Explain

number_of_movies	
	bigint
1	14

Result 2

Data Output Explain Messages Notifications

	customer_id	store_id	first_name	last_name	email
1	61	2	KATHERINE	RIVERA	KATHE
2	339	2	WALTER	PERRYMAN	WALTE
3	535	1	JAVIER	ELROD	JAVIER
4	46	2	CATHERINE	CAMPBELL	CATHE
5	149	1	VALERIE	BLACK	VALERI

Result 3

Data Output

count	
	bigint
1	27

# Solution 1

1. How many movies are there that contain 'Saga' in the description and where the title starts either with 'A' or ends with 'R'?  
Use the alias 'no\_of\_movies'.

```
SELECT
COUNT(*) AS number_of_movie
FROM film
WHERE description LIKE '%Saga%'
AND (title LIKE 'A%'
OR title LIKE '%R')
```

# Solution 2

2. Create a list of all customers where the first name contains 'ER' and has an 'A' as the second letter. Order the results by the last name descendingly.

```
SELECT  
*  
FROM customer  
WHERE first_name LIKE '%ER%'  
AND first_name LIKE '_A%'  
ORDER BY last_name DESC
```

# Solution 3

3. How many payments are there where the amount is either 0 or between 3.99 and 7.99 and in the same time have happened on 2020-05-01.

```
SELECT  
COUNT(*)  
FROM payment  
WHERE (amount = 0  
OR amount BETWEEN 3.99 AND 7.99)  
AND payment_date BETWEEN '2020-05-01' AND '2020-05-02'
```

## Solution 2

4. How many distinct last names of the customers are there?

```
SELECT  
COUNT(DISTINCT last_name)  
FROM customer
```

# Day 3

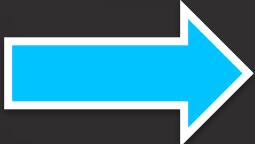


# Aggregation functions

- ✓ Aggregate values in multiple rows to one value

Data Output Explain	
	amount numeric (5,2)
1	1.99
2	0.99
3	6.99
4	0.99
5	4.99
6	2.99

SUM



Data Output Explain	
	sum numeric
1	67416.51

# Aggregation functions

- ✓ Aggregate values in multiple rows to one value

Data Output Explain	
	amount numeric (5,2)
1	1.99
2	0.99
3	6.99
4	0.99
5	4.99
6	2.99

AVG



Data Output Explain	
	avg numeric
1	4.20

# Most common aggregation functions

SUM()

AVG()

MIN()

MAX()

COUNT()

# SYNTAX

```
SELECT  
SUM(amount)  
FROM payment
```

Data Output		Expl
	sum numeric	🔒
1	67416.51	

# SYNTAX

```
SELECT  
COUNT(*)  
FROM payment
```

# What we can't do...

```
SELECT  
SUM(amount)  
payment_id  
FROM payment
```

Data Output Expl

	sum	
	numeric	🔒
1	67416.51	

# No mixing possible!

```
SELECT  
SUM(amount)  
payment_id ←  
FROM payment
```

Only possible with  
grouping!

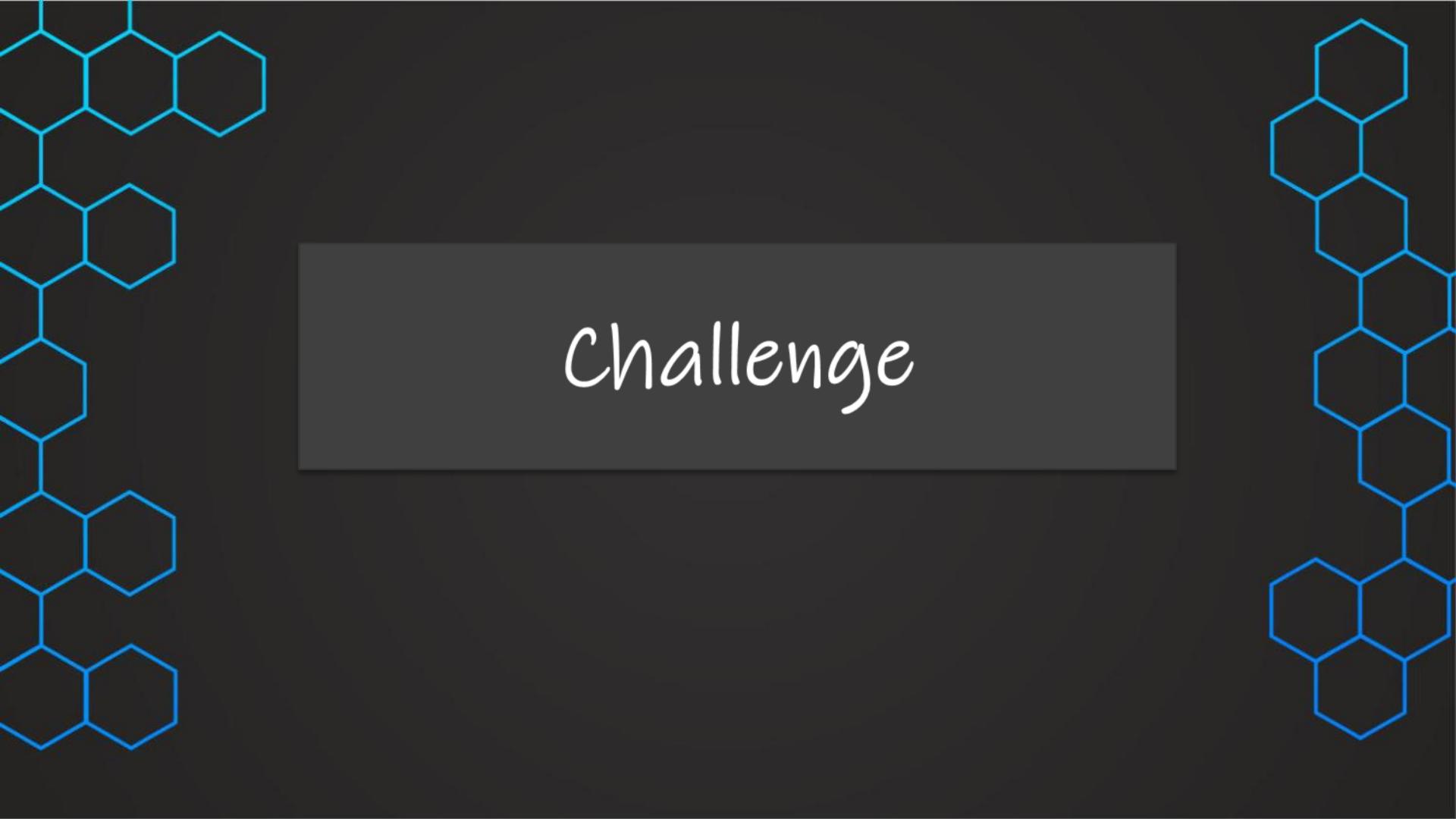
	sum	
	numeric	🔒
1	67416.51	

# Multiple aggregations is possible!

```
SELECT  
SUM(amount) ,  
COUNT(*) ,  
AVG(amount)  
FROM payment
```

Data Output Explain Messages

	sum numeric	count bigint	avg numeric
1	67416.51	16049	4.20



Challenge

# Challenge

Your manager wants to know which of the two employees (staff\_id) is responsible for more payments?

Which of the two is responsible for a higher overall payment amount?

How do these amounts change if we don't consider amounts equal to 0?

**Write two SQL queries to get the answers!**

Result

Data Output		Explain	Messages
	staff_id smallint lock	sum numeric lock	count bigint lock
1	2	33927.04	7992
2	1	33489.47	8057

Data Output		Explain	Messages
	staff_id smallint lock	sum numeric lock	count bigint lock
1	2	33927.04	7983
2	1	33489.47	8042

# Solution

```
SELECT  
    MIN(replacement_cost) ,  
    MAX(replacement_cost) ,  
    ROUND(AVG(replacement_cost),2) AS AVG,  
    SUM(replacement_cost)  
FROM film
```

# GROUP BY

✓ Used to GROUP aggregations BY specific columns

	customer_id	amount
1	269	1.99
2	269	0.99
3	269	6.99
4	269	0.99
5	269	4.99
6	269	2.99
7	270	1.99
8	270	4.99



	customer_id	sum
1	1	118.68
2	2	128.73
3	3	135.74
4	4	81.78
5	5	144.62

# SYNTAX

```
SELECT  
customer_id,  
SUM(amount)  
FROM payment  
GROUP BY customer_id
```

	customer_id	sum
1	1	118.68
2	2	128.73
3	3	135.74
4	4	81.78
5	5	144.62

# SYNTAX

```
SELECT  
customer_id,  
SUM(amount)  
FROM payment  
WHERE customer_id >3  
GROUP BY customer_id
```

	customer_id	sum
1	1	118.68
2	2	128.73
3	3	135.74
4	4	81.78
5	5	144.62

# SYNTAX

```
SELECT  
customer_id,  
SUM(amount)  
FROM payment  
WHERE customer_id >3  
GROUP BY customer_id  
ORDER BY customer_id
```

	customer_id	sum
1	1	118.68
2	2	128.73
3	3	135.74
4	4	81.78
5	5	144.62

# SYNTAX

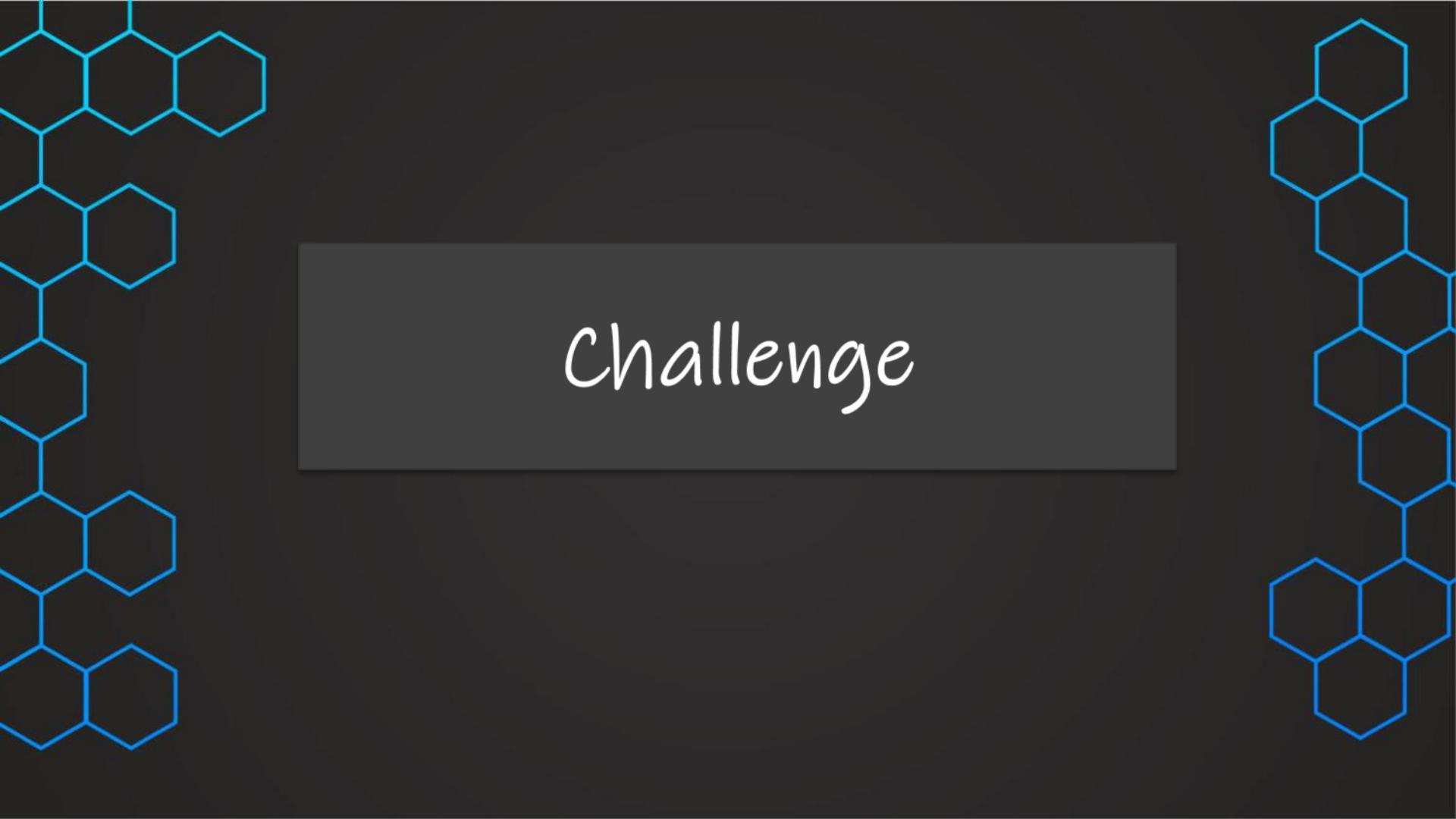
```
SELECT  
customer_id,  
SUM(amount)  
FROM payment  
GROUP BY customer_id
```

Every column:  
In GROUP BY or  
in aggregate functions

# SYNTAX

```
SELECT  
customer_id,  
SUM(amount)  
FROM payment  
GROUP BY customer_id
```

Every column:  
In GROUP BY or  
in aggregate functions



Challenge

# Challenge

There are two competitions between the two employees.

Which employee had the highest sales amount in a single day?

Which employee had the most sales in a single day (not counting payments with amount = 0)?

**Write two SQL queries to get the answers!**

Result

	date	staff_id	sum	count
	date	smallint	numeric	bigint
1	2020-04-30	2	2866.42	658
2	2020-04-30	1	2736.75	625
3	2020-03-21	2	1505.52	348



Challenge

# Challenge

Your manager wants to get a better understanding of the films.

That's why you are asked to write a query to see the

- Minimum
- Maximum
- Average (rounded)
- Sum

of the replacement cost of the films.

**Write a SQL query to get the answers!**

Result

Data Output		Explain	Messages	Notifications
	min numeric	max numeric	avg numeric	sum numeric
1	9.99	29.99	19.98	19984.00

# HAVING

✓ Used to FILTER Groupings BY aggregations

HAVING  
COUNT(\*)>400



	staff_id	date	sum	count
	smallint	date	numeric	bigint
1	2	2020-04-30	2866.42	658
2	1	2020-04-30	2736.75	625
3	2	2020-03-21	1505.52	348
4	1	2020-03-01	143	
5	2	2020-03-19	140	

	staff_id	date	sum	count
	smallint	date	numeric	bigint
1	2	2020-04-30	2866.42	658
2	1	2020-04-30	2736.75	625

Note!

HAVING can only be used  
with GROUP BY!

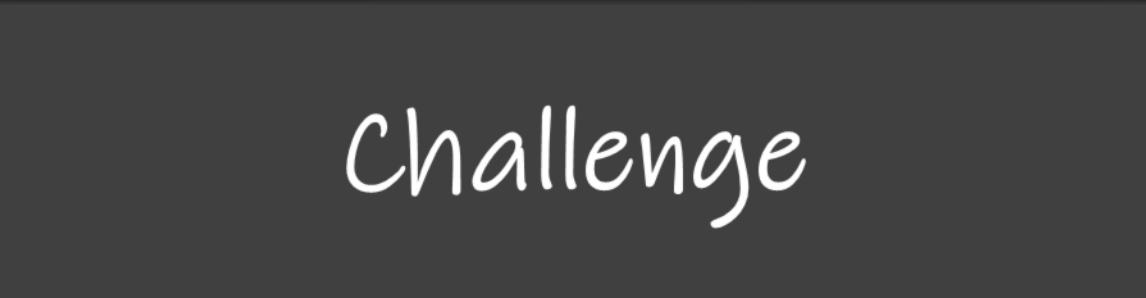
# SYNTAX

```
SELECT  
customer_id,  
SUM(amount)  
FROM payment  
GROUP BY customer_id  
HAVING SUM(amount)>200
```

Data Output			Explain	Mes
	customer_id:	sum		
	smallint	numeric	lock	lock
1		526		
2		148		
		221.55		
		216.54		

# Solution

```
SELECT  
    MIN(replacement_cost) ,  
    MAX(replacement_cost) ,  
    ROUND(AVG(replacement_cost) ,2) AS AVG,  
    SUM(replacement_cost)  
FROM film
```



Challenge

# Challenge

In 2020, April 28, 29 and 30 were days with very high revenue. That's why we want to focus in this task only on these days (filter accordingly).

Find out what is the **average payment amount grouped by customer and day** – consider only the days/customers with more than 1 payment (per customer and day).  
Order by the average amount in a descending order.

**Write a SQL query to find out!**

Result

Data Output		Explain	Messages	Notifications
	customer_id	date	avg_amount	count
1	459	2020-04-29	10.49	2
2	443	2020-04-28	9.49	2
3	510	2020-04-28	9.49	2

# Day 4





Challenge

# Challenge

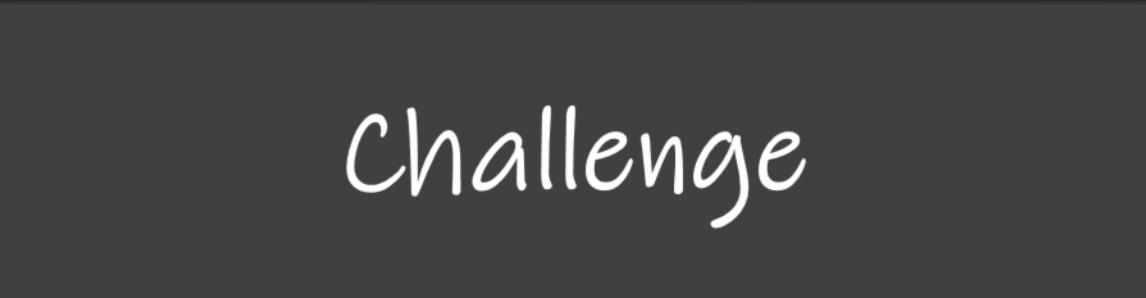
In the email system there was a problem with names where either the first name or the last name is more than 10 characters long.

Find these customers and output the list of these first and last names in all lower case.

**Write a SQL query to find out!**

Result

	Data Output	Explain	Messages	Notifications
	lower text	lower text	lower text	lower text
1	william	satterfield	william.satterfield@sakilacustomer.org	
2	christopher	greco	christopher.greco@sakilacustomer.org	
3	henry	billingsley	henry.billingsley@sakilacustomer.org	
4	elijah	spencer	elijah.spencer@sakilacustomer.org	



Challenge

# Challenge

In this challenge you have only the email address and the last name of the customers.

	email text	last_name text
1	MARY.SMITH@sakilacustomer.org	SMITH
2	PATRICIA.JOHNSON@sakilacustomer.org	JOHNSON
3	LINDA.WILLIAMS@sakilacustomer.org	WILLIAMS
4	BARBARA.JONES@sakilacustomer.org	JONES

You need to extract the first name from the email address and concatenate it with the last name. It should be in the form: "Last name, First name".

**Write a SQL query to find out!**

Result

	?column? text
1	SMITH, MARY
2	JOHNSON, PATRICIA
3	WILLIAMS, LINDA



Challenge

# Challenge

Extract the last 5 characters of the email address first.

The email address always ends with '.org'.

How can you extract just the dot '.' from the email address?

**Write a SQL query to find out!**

Result

Data Output	
	right text
1	r.org
2	r.org
3	r.org

Data Output	
	left text
1	.
2	.
3	.

# SUBSTRING

✓ Used to EXTRACT a SUBSTRING from a string

	Data Output	Explain	Messages	Notifications
	email text			
1	MARY.SMITH@sakilacustomer.org			
2	PATRICIA.JOHNSON@sakilacustomer.org			
3	LINDA.WILLIAMS@sakilacustomer.org			

SUBSTRING



substring	text
SMITH	
JOHNSON	
WILLIAMS	

# SYNTAX

SUBSTRING (string from start [for length] )

column / string  
that we want to extract from

Position,  
Where to start from?

Length,  
How many characters?

# SYNTAX

SUBSTRING (email from start [for length] )

column / string  
that we want to extract from

Position,  
Where to start from?

Length,  
How many characters?

# SYNTAX

```
SUBSTRING (email from 2 [for length] )
```

column / string  
that we want to extract from

Position,  
Where to start from?

Length,  
How many characters?

# SYNTAX

```
SUBSTRING (email from 2 for 3 )
```

column / string  
that we want to extract from

Position,  
Where to start from?

Length,  
How many characters?

# SYNTAX

Length,  
How many characters?

SUBSTRING (email from 2 for 3 )

column / string

that w

Position,

t from?

Data Output

Explain

Messages

Notifications

email  
text

substring  
text

MARY.SMITH@sakilacustomer.org

ARY

PATRICIA.JOHNSON@sakilacustomer.org

CTR

LINDA.WILLIAMS@sakilacustomer.org

IND

# SYNTAX

SUBSTRING (email from 2 )

column / string  
that we want to extract from

Position,  
Where to start from?

	email text	substring text
1	MARY.SMITH@sakilacustomer.org	ARY.SMITH@sakilacustomer.org
2	PATRICIA.JOHNSON@sakilacustomer.org	ATRICIA.JOHNSON@sakilacustomer.org
3	LINDA.WILLIAMS@sakilacustomer.org	INDA.WILLIAMS@sakilacustomer.org

# SYNTAX

SUBSTRING (email from POSITION ('.' in email) for 3 )

Length,  
How many characters?

that we can use to extract a portion of a string.

column / string

Position,

Data Output

Explain

Messages

Notifications

email  
text

substring  
text

MARY.SMITH@sakilacustomer.org

.SM

PATRICIA.JOHNSON@sakilacustomer.org

.JO

LINDA.WILLIAMS@sakilacustomer.org

.WI

# SYNTAX

Length,  
How many characters?

SUBSTRING (email from POSITION ('.' in email)+1 for 3 )

column / string

that w

Position,

t from?

	email text	substring text
1	MARY.SMITH@sakilacustomer.org	SMI
2	PATRICIA.JOHNSON@sakilacustomer.org	JOH
3	LINDA.WILLIAMS@sakilacustomer.org	WIL

# SYNTAX

SUBSTRING (string from start [for length])

column / string  
that we want to extract from

Position,  
Where to start from?

Length,  
How many characters?



Challenge

# Challenge

You need to create an anonymized form of the email addresses in the following way:

1	M***.S***@sakilacustomer.org
2	P***.J***@sakilacustomer.org

In a second query create an anonymized form of the email addresses in the following way:

1	***Y.S***@sakilacustomer.org
2	***A.J***@sakilacustomer.org

**Write a SQL query to find out!**

# EXTRACT

✓ Used to EXTRACT parts of timestamp/date

rental_date	timestamp with time zone
2005-05-24 23:54:33+02	
2005-05-25 00:03:39+02	
2005-05-25 00:04:41+02	
2005-05-25 00:05:21+02	

EXTRACT (day)



extract	numeric
	24
	25
	25
	25

# EXTRACT

✓ Used to EXTRACT parts of timestamp/date

rental_date	timestamp with time zone
2005-05-24 23:54:33+02	
2005-05-25 00:03:39+02	
2005-05-25 00:04:41+02	
2005-05-25 00:05:21+02	

EXTRACT (seconds)



extract	numeric
33.000000	
39.000000	
41.000000	
21.000000	

# Date/time types

date	Just date without time	'2022-11-28'
time (with/without time zone)	Just time without date	'01:02:03.678'
timestamp (with/without time zone)	Date and time	'2022-11-28 01:02:03.678+02'
intervals	Time interval	'3 days 01:02:03.678'

# SYNTAX

EXTRACT (*field* from *date/time/interval*)

Part of date/time

Date/time  
that we want to extract from

# EXTRACT

Usually singular

Useful when  
creating new  
tables

Field	Extract from timestamp/date
CENTURY	century
DAY	day of month (1-31)
DECADE	decade that is year divided by 10
DOW	day of week Sunday (0) to Saturday (6)
DOY	day of year that ranges from 1 to 366
EPOCH	number of seconds since 1970-01-01 00:00:00 UTC
HOUR	hour (0-23)
ISODOW	day of week based on ISO 8601 Monday (1) to Sunday (7)
ISOYEAR	ISO 8601 week number of year
MICROSECONDS	seconds field, including fractional parts, multiplied by 1000000
MILLENNIUM	millennium
MILLISECONDS	seconds field, including fractional parts, multiplied by 1000
MINUTE	minute (0-59)
MONTH	month (1-12)
QUARTER	quarter of year
SECOND	second
TIMEZONE	timezone offset from UTC, measured in seconds
TIMEZONE_HOUR	hour component of time zone offset
TIMEZONE_MINUTE	minute component of time zone offset
WEEK	number of ISO 8601 week-numbering week of year
YEAR	year

# EXTRACT

Usually singular

Useful when  
creating new  
tables

Field	Extract from timestamp/date
CENTURY	century
DAY	day of month (1-31)
DECADE	decade that is year divided by 10
DOW	day of week Sunday (0) to Saturday (6)
DOY	day of year that ranges from 1 to 366
EPOCH	number of seconds since 1970-01-01 00:00:00 UTC
HOUR	hour (0-23)
ISODOW	day of week based on ISO 8601 Monday (1) to Sunday (7)
ISOYEAR	ISO 8601 week number of year
MICROSECONDS	seconds field, including fractional parts, multiplied by 1000000
MILLENNIUM	millennium
MILLISECONDS	seconds field, including fractional parts, multiplied by 1000
MINUTE	minute (0-59)
MONTH	month (1-12)
QUARTER	quarter of year
SECOND	second
TIMEZONE	timezone offset from UTC, measured in seconds
TIMEZONE_HOUR	hour component of time zone offset
TIMEZONE_MINUTE	minute component of time zone offset
WEEK	number of ISO 8601 week-numbering week of year
YEAR	year

# Challenge

You need to analyze the payments and find out the following:

- What's the month with the highest total payment amount?
- What's the day of week with the highest total payment amount? (0 is Sunday)
- What's the highest amount one customer has spent in a week?

**Write a SQL query to find out!**

Result

month	total_payment_amount
numeric	numeric
1	28327.02
2	23886.56

day_of_week	total_payment_amount
numeric	numeric
1	12796.08
2	12132.12

week	customer_id	total_payment_amount
numeric	smallint	numeric
1	459	73.88
2	21	72.86
3	2	65.88

# TO\_CHAR

✓ Used to get custom formats timestamp/date/numbers

rental_date	timestamp with time zone
2005-05-24 23:54:33+02	
2005-05-25 00:03:39+02	
2005-05-25 00:04:41+02	
2005-05-25 00:05:21+02	

TO\_CHAR (YYYY-MM)



	to_char	text
1		2005-05
2		2005-05
3		2005-05
4		2005-05

# TO\_CHAR

✓ Used to get custom formats timestamp/date/numbers

rental_date
timestamp with time zone
2005-05-24 23:54:33+02
2005-05-25 00:03:39+02
2005-05-25 00:04:41+02
2005-05-25 00:05:21+02

TO\_CHAR (Month)



	to_char
	text
1	May
2	May
3	May
4	May

# SYNTAX

TO\_CHAR (date/time/interval, format)

date/time/interval/number

Format

# SYNTAX

TO\_CHAR (rental\_date, format)

date/time/interval/number

Format

# SYNTAX

```
TO_CHAR (rental_date, 'MM-YYYY')
```

date/time/interval/number

Format

ym	text
05-2020	
03-2020	
04-2020	

# Challenge

You need to sum payments and group in the following formats:

total_amount numeric	day text
62.86	Fri, 24/01/2020
70.81	Fri, 14/02/2020

total_amount numeric	day text
1	746.62
2	4824.43

total_amount numeric	day text
1	537.14
2	59.90

**Write a SQL query to find out!**

Result

month numeric	total_payment_amount numeric
1	28327.02
2	23886.56

day_of_week numeric	total_payment_amount numeric
1	12796.08
2	12132.12

week numeric	customer_id smallint	total_payment_amount numeric
1	18	459
2	12	21
3	18	2

# Challenge

You need to create a list for the suppcity team of all rental durations of customer with customer\_id 35.

Also you need to find out for the suppcity team which customer has the longest average rental duration?

**Write a SQL query to find out!**

Result

customer_id	rental_duration
1	35 4 days 20:59:00
2	35 8 days 18:10:00
3	35 5 days 01:12:00

customer_id	avg
1	315 6 days 14:13:22.5
2	187 5 days 34:58:38.571428
3	321 5 days 32:56:32.727273

# Day 5



# Mathematical functions and operators

Operator	Description	Example	Result
<code>+</code>	addition	$4 + 3$	7
<code>-</code>	subtraction	$5 - 3$	2
<code>*</code>	multiplication	$4 * 2$	8
<code>/</code>	division (integer division truncates the result)	$8 / 4$	2
<code>%</code>	modulo (remainder)	$10 \% 4$	2
<code>^</code>	exponentiation	$2 ^ 3$	8

# SYNTAX

```
SUM (replacement_cost) * 2
```

```
SUM (replacement_cost) + 1
```

```
SUM (replacement_cost) / SUM (rental_rate)*100
```

# Mathematical functions and operators

Function	Description	Example	Result
<b>abs(x)</b>	absolute value	abs(-7)	7
<b>round(x,d)</b>	round x to d decimal places	round(4.3543)	4.35
<b>ceiling(x)</b>	round up to integer	ceiling(4.3543)	5
<b>floor(x)</b>	round down to integer	floor(4.3543)	4

# Challenge

Your manager is thinking about increasing the prices for films that are more expensive to replace.

For that reason, you should create a list of the films including the relation of rental rate / replacement cost where the rental rate is less than 4% of the replacement cost.

Create a list of that film\_ids together with the percentage rounded to 2 decimal places. For example 3.54 (=3.54%).

Result

	film_id [PK] integer	percentage numeric
1	417	3.30
2	663	3.30

X

	film_id [PK] integer	percentage numeric
1	417	3.3000
2	663	3.3000

X

	film_id [PK] integer	percentage numeric
1	395	3.00
2	196	3.00

# CASE

Like IF/THEN statement:

Goes through a set of conditions  
returns a value if a condition is met

# SYNTAX

Start of CASE statement

CASE

WHEN condition1 THEN result1

WHEN condition2 THEN result2

WHEN conditionN THEN resultN

ELSE result

END

End of CASE statement

Conditions & results

# SYNTAX

```
SELECT  
amount,  
CASE  
WHEN amount < 2 THEN 'low amount'  
WHEN amount < 5 THEN 'medium amount'  
ELSE 'high amount'  
END  
FROM payment
```

amount numeric (5,2)	case text
1.99	low amout
0.99	low amout
6.99	high amount
0.99	low amout
4.99	medium amount

# SYNTAX

```
SELECT  
TO_CHAR(book_date, 'Dy') ,  
TO_CHAR(book_date, 'Mon') ,  
CASE  
WHEN TO_CHAR(book_date, 'Dy')='Mon' THEN 'Monday special'  
WHEN TO_CHAR(book_date, 'Mon')='Jul' THEN 'July special'  
END  
FROM bookings
```

Result of first true condition!

Mon	Aug	Monday special
Sat	Jul	July special
Tue	Jul	July special
Mon	Jul	Monday special

# SYNTAX

```
SELECT  
TO_CHAR(book_date, 'Dy') ,  
TO_CHAR(book_date, 'Mon') ,  
CASE  
WHEN TO_CHAR(book_date, 'Dy')='Mon' THEN 'Monday special'  
WHEN TO_CHAR(book_date, 'Mon')='Jul' THEN 'July special'  
END  
FROM bookings
```

No condition met => [null]

Wed	Jul	July special
Fri	Jul	July special
Tue	Aug	[null]
Thu	Aug	[null]
Mon	Aug	Monday special

# SYNTAX

```
SELECT  
TO_CHAR(book_date, 'Dy') ,  
TO_CHAR(book_date, 'Mon') ,  
CASE  
WHEN TO_CHAR(book_date, 'Dy')='Mon' THEN 'Monday special'  
WHEN TO_CHAR(book_date, 'Mon')='Jul' THEN 'July special'  
ELSE 'no special'  
END  
FROM bookings
```

*ELSE => result if no condition is met*

Wed	Jul	July special
Fri	Jul	July special
Tue	Aug	no special
Thu	Aug	no special
Mon	Aug	Monday special

# SYNTAX

```
SELECT  
total_amount,  
TO_CHAR(book_date, 'Dy'),  
CASE  
WHEN TO_CHAR(book_date, 'Dy')='Mon' THEN 'Monday special'  
WHEN total_amount < 30000 THEN 'Special deal'  
ELSE 'no special at all'  
END  
FROM bookings
```

# SYNTAX

```
SELECT  
total_amount,  
TO_CHAR(book_date, 'Dy'),  
CASE  
WHEN TO_CHAR(book_date, 'Dy')='Mon' THEN 'Monday special'  
WHEN total_amount*1.4 < 30000 THEN 'Special deal'  
ELSE 'no special at all'  
END  
FROM bookings
```

total_amount	to_char	case
numeric (10,2)	text	text
265700.00	Wed	no special at all
37900.00	Fri	no special at all
18100.00	Tue	Special deal
131800.00	Thu	no special at all
23600.00	Mon	Monday special

# SYNTAX

```
SELECT  
total_amount,  
TO_CHAR(book_date,'Dy'),  
CASE  
    WHEN TO_CHAR(book_date,'Dy')='Mon' THEN 'Monday special'  
    WHEN total_amount < 30000 THEN 'Special deal'  
    ELSE 'no special at all'  
END  
FROM bookings
```

# Challenge

You need to find out how many tickets you have sold in the following categories:

- Low price ticket: total\_amount < 20,000
- Mid price ticket: total\_amount between 20,000 and 150,000
- High price ticket: total\_amount >= 150,000

How many high price tickets has the company sold?

Result

Data Output		Explain	Message
	count bigint	ticket_price text	
1	205036	mid price ticket	
2	30012	high price ticket	
3	27740	low price ticket	

# Challenge

You need to find out how many flights have departed in the following seasons:

- Winter: December, January, Februar
- Spring: March, April, May
- Summer: June, July, August
- Fall: September, October, November

Result

Data Output Explain

	flights bigint	season text
1	7596	Fall
2	25525	Summer

# Challenge

You want to create a tier list in the following way:

1. Rating is 'PG' or 'PG-13' or length is more than 210 min:  
'Great rating or long (tier 1)'
2. Description contains 'Drama' and length is more than 90min:  
'Long drama (tier 2)'
3. Description contains 'Drama' and length is not more than 90min:  
'Shcity drama (tier 3)'
4. Rental\_rate less than \$1:  
'Very cheap (tier 4)'

If one movie can be in multiple categories it gets the higher tier assigned.  
How can you filter to only those movies that appear in one of these 4 tiers?

Result

	title text	case text
1	ACADEMY DINOSAUR	Great rating or very long (tier 1)
2	AGENT TRUMAN	Great rating or very long (tier 1)
3	AIRPLANE SIERRA	Great rating or very long (tier 1)
4	ALABAMA DEVIL	Great rating or very long (tier 1)
5	ALAMO VIDEOTAPE	Very cheap (tier 4)

# COALESCE

✓ Returns first value of a list of values which is not null

COALESCE(actual\_arrival, schedule\_arrival)

actual_arrival	scheduled_arrival
timestamp with time zone	timestamp with time zone
2017-08-05 19:01:00+02	2017-08-05 19:00:00+02
2017-08-05 09:34:00+02	2017-08-05 09:30:00+02
[null]	2017-09-09 12:20:00+02



coalesce
timestamp with time zone
2017-08-05 19:01:00+02
2017-08-05 09:34:00+02
2017-09-09 12:20:00+02

# SYNTAX

COALESCE (actual\_arrival, scheduled\_arrival)

# SYNTAX

```
COALESCE (actual_arrival, '1970-01-01 0:00')
```

actual_arrival timestamp with time zone	scheduled_arrival timestamp with time zone	coalesce timestamp with time zone
[null]	2017-09-10 13:55:00+02	1970-01-01 00:00:00+01

# CAST

✓ Changes the data type of a value

CAST TO TEXT

scheduled_arrival	timestamp with time zone
2017-09-10 13:55:00+02	
2017-08-25 16:35:00+02	



scheduled_arrival	character varying
2017-09-10 13:55:00+02	
2017-08-25 16:35:00+02	

# SYNTAX

CAST (*value/column AS data type*)

# SYNTAX

```
CAST (scheduled_arrival AS data type)
```

# SYNTAX

```
CAST (scheduled_arrival AS VARCHAR)
```

scheduled_arrival	character varying	🔒
2017-09-10 13:55:00+02		
2017-08-25 16:35:00+02		

# SYNTAX

CAST (scheduled\_arrival AS DATE)

scheduled_arrival	scheduled_arrival
timestamp with time zone	date
2017-09-10 13:55:00+02	2017-09-10
2017-08-25 16:35:00+02	2017-08-25
2017-09-05 13:15:00+02	2017-09-05

# REPLACE

✓ Replaces text from a string in a column with another text

REPLACE 'PG' with 'FL'

flight_no
character (6)
PG0134
PG0052
PG0561



replace
text
FL0134
FL0052
FL0561

# REPLACE

✓ Replaces text from a string in a column with another text

REPLACE 'PG' with "

flight_no
character (6)
PG0134
PG0052
PG0561



replace
text
0134
0052
0561

# SYNTAX

```
REPLACE (column, old_text, new_text)
```

# SYNTAX

```
REPLACE (flight_no, 'PG', 'FL')
```

replace   
text

FL0134

FL0052

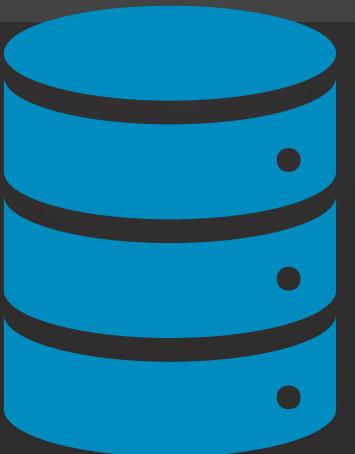
FL0561

# SYNTAX

```
REPLACE (flight_no, 'PG', '')
```

replace	text
0134	
0052	
0561	

# Day 6



# What are JOINS?

# JOINS

✓ Combine information from multiple tables in one query

	payment_id	customer_id	staff_id	rental_id	amount	payment_date
1	1605	269	2	7	1.99	2020-01-24 22:40:19.996577+01
2	1605	269	1	98	0.99	2020-01-25 16:16:50.996577+01
3	1605	269	2	678	6.99	2020-01-28 22:44:14.996577+01

customer_id	store_id	first_name	last_name	email	payment_id	customer_id	staff_id	rental_id	amount	payment_date	first_name	last_name
269	1	CASSANDRA	WALTERS	CASSANDRA.WALTERS@sakilacustomer.org	7	1605	269	2	1.99	2020-01-24 22:40:19.996577+01	CASSANDRA	WALTERS
					98				0.99	2020-01-25 16:16:50.996577+01	CASSANDRA	WALTERS
					678				6.99	2020-01-28 22:44:14.996577+01	CASSANDRA	WALTERS

	customer_id	store_id	first_name	last_name	email
1	1	1	MARY	SMITH	MARY.SMITH@sakilacustomer.org
2	2	1	PATRICIA	JOHNSON	PATRICIA.JOHNSON@sakilacustomer.org
3	3	1	LINDA	WILLIAMS	LINDA.WILLIAMS@sakilacustomer.org

# What are Joins

- ✓ How Joins work and how we use them practically

**INNER JOIN**

**OUTER JOIN**

**LEFT JOIN**

**RIGHT JOIN**

# What are Joins

- ✓ **Very important technique in SQL**
- ✓ **Can seem a bit complicated at first**
- ✓ **Theory + practice**
- ✓ **Many examples + practical tips + challenges**

# Inner Join - Theory

# INNER JOIN

- ✓ Inner Join easiest join type
- ✓ Helps to understand the general concept of joins

# INNER JOIN

**sales table**

employee	city	sales
Sandra	Frankfurt	500
Sabine	Munich	300
Peter	Hamburg	200
Manuel	Hamburg	400
Michael	Munich	100
Frank	Frankfurt	100

**bonus table**

employee	bonus
Sandra	YES
Sabine	YES
Peter	NO
Manuel	YES
Simon	NO

✓ Combine the two tables in one query

✓ One common column – join column

# INNER JOIN

**sales table**

employee	city	sales
Sandra	Frankfurt	500
Sabine	Munich	300
Peter	Hamburg	200
Manuel	Hamburg	400
Michael	Munich	100
Frank	Frankfurt	100

**bonus table**

employee	bonus
Sandra	YES
Sabine	YES
Peter	NO
Manuel	YES
Simon	NO

Do not appear in the bonus table

# INNER JOIN

**sales table**

employee	city	sales
Sandra	Frankfurt	500
Sabine	Munich	300
Peter	Hamburg	200
Manuel	Hamburg	400
Michael	Munich	100
Frank	Frankfurt	100

Do not appear in the bonus table

**bonus table**

employee	bonus
Sandra	YES
Sabine	YES
Peter	NO
Manuel	YES
Simon	NO

Do not appear in the sales table

# INNER JOIN

**sales table**

employee	city	sales
Sandra	Frankfurt	500
Sabine	Munich	300
Peter	Hamburg	200
Manuel	Hamburg	400
Michael	Munich	100
Frank	Frankfurt	100

**bonus table**

employee	bonus
Sandra	YES
Sabine	YES
Peter	NO
Manuel	YES
Simon	NO

Do not appear in the bonus table

Do not appear in the sales table

# INNER JOIN

**sales table**

employee	city	sales
Sandra	Frankfurt	500
Sabine	Munich	300
Peter	Hamburg	200
Manuel	Hamburg	400
Michael	Munich	100
Frank	Frankfurt	100

**bonus table**

employee	bonus
Sandra	YES
Sabine	YES
Peter	NO
Manuel	YES
Simon	NO

Do not appear in the bonus table

Do not appear in the sales table

✓ INNER JOIN: Only rows appear in both tables

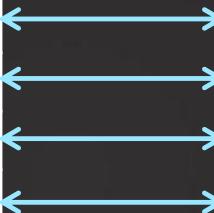
# INNER JOIN

**sales table**

employee	city	sales
Sandra	Frankfurt	500
Sabine	Munich	300
Peter	Hamburg	200
Manuel	Hamburg	400
Michael	Munich	100
Frank	Frankfurt	100

**bonus table**

employee	bonus
Sandra	YES
Sabine	YES
Peter	NO
Manuel	YES
Simon	NO



employee	city	sales	bonus

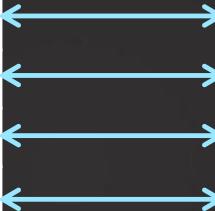
# INNER JOIN

**sales table**

employee	city	sales
Sandra	Frankfurt	500
Sabine	Munich	300
Peter	Hamburg	200
Manuel	Hamburg	400
Michael	Munich	100
Frank	Frankfurt	100

**bonus table**

employee	bonus
Sandra	YES
Sabine	YES
Peter	NO
Manuel	YES
Simon	NO



employee	city	sales	bonus
Sandra	Frankfurt	500	

# INNER JOIN

**sales table**

employee	city	sales
Sandra	Frankfurt	500
Sabine	Munich	300
Peter	Hamburg	200
Manuel	Hamburg	400
Michael	Munich	100
Frank	Frankfurt	100

**bonus table**

employee	bonus
Sandra	YES
Sabine	YES
Peter	NO
Manuel	YES
Simon	NO

employee	city	sales	bonus
Sandra	Frankfurt	500	YES

# INNER JOIN

**sales table**

employee	city	sales
Sandra	Frankfurt	500
Sabine	Munich	300
Peter	Hamburg	200
Manuel	Hamburg	400
Michael	Munich	100
Frank	Frankfurt	100

**bonus table**

employee	bonus
Sandra	YES
Sabine	YES
Peter	NO
Manuel	YES
Simon	NO

employee	city	sales	bonus
Sandra	Frankfurt	500	YES
Sabine	Munich	300	YES
Peter	Hamburg	200	NO
Manuel	Hamburg	400	YES

# INNER JOIN

Table A

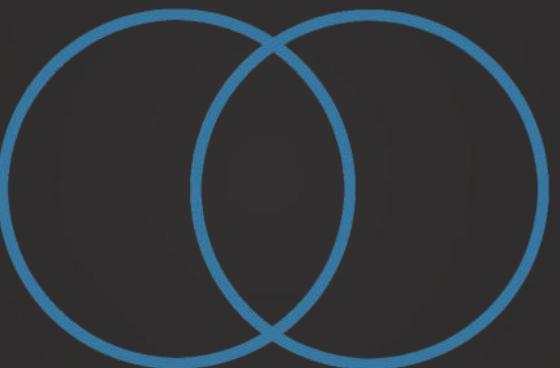


Table B

# INNER JOIN

Table A

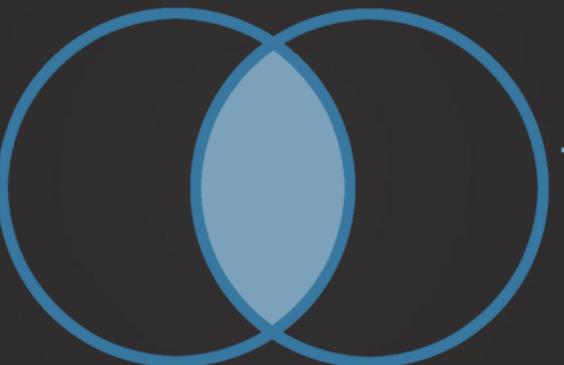


Table B

# INNER JOIN

employee	city	sales
Sandra	Frankfurt	500
Sabine	Munich	300
Peter	Hamburg	200
Manuel	Hamburg	400
Michael	Munich	100
Frank	Frankfurt	100

employee	bonus
Sandra	YES
Sabine	YES
Peter	NO
Manuel	YES
Simon	NO



employee	city	sales	bonus
Sandra	Frankfurt	500	YES
Sabine	Munich	300	YES
Peter	Hamburg	200	NO
Manuel	Hamburg	400	YES

Table A

Table B

# INNER JOIN

employee	city	sales
Sandra	Frankfurt	500
Sabine	Munich	300
Peter	Hamburg	200
Peter	Munich	250
Manuel	Hamburg	400
Michael	Munich	100
Frank	Frankfurt	100
employee	bonus	
Sandra	YES	
Sabine	YES	
Peter	NO	
Manuel	YES	
Simon	NO	



employee	city	sales	bonus
Sandra	Frankfurt	500	YES
Sabine	Munich	300	YES
Peter	Hamburg	200	NO
Manuel	Hamburg	400	YES

Table A

Table B

# INNER JOIN

Table A

employee	city	sales
Sandra	Frankfurt	500
Sabine	Munich	300
Peter	Hamburg	200
Peter	Munich	250
Manuel	Hamburg	400
Michael	Munich	100
Frank	Frankfurt	100
employee	bonus	
Sandra	YES	
Sabine	YES	
Peter	NO	
Manuel	YES	
Simon	NO	

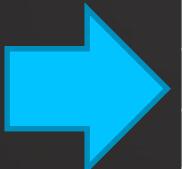


Table B

employee	city	sales	bonus
Sandra	Frankfurt	500	YES
Sabine	Munich	300	YES
Peter	Hamburg	200	NO
Peter	Munich	250	NO
Manuel	Hamburg	400	YES

# INNER JOIN

employee	city	sales
Sandra	Frankfurt	500
Sabine	Munich	300
Peter	Hamburg	200
Peter	Munich	250
Manuel	Hamburg	400
Michael	Munich	100
Frank	Munich	100

employee	bonus
Sandra	YES
Sabine	YES
Peter	NO
Manuel	YES
Simon	NO



Table A

Table B

employee	-	-	bonus
Sandra	-	-	YES
Sabine	-	-	YES
Peter	-	-	NO
Peter	-	-	NO
Manuel	-	-	YES

# INNER JOIN

Table A

employee	city	sales
Sandra	Frankfurt	500
Sabine	Munich	300
Peter	Hamburg	200
Peter	Munich	250
Manuel	Hamburg	400
Michael	Munich	100
Frank	Frankfurt	100

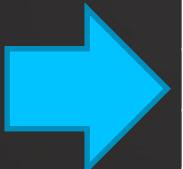


Table B

employee	city	sales	bonus
Sandra	Frankfurt	500	YES
Sabine	Munich	300	YES
Peter	Hamburg	200	NO
Peter	Munich	250	NO
Manuel	Hamburg	400	YES

# SYNTAX

Table A      Table B



```
SELECT * FROM TableA  
INNER JOIN TableB  
ON TableA.employee = TableB.employee
```

# SYNTAX

Table A      Table B



```
SELECT * FROM TableB  
INNER JOIN TableA  
ON TableA.employee = TableB.employee
```

# SYNTAX

Table A      Table B

```
SELECT * FROM TableA AS A  
INNER JOIN TableB AS B  
ON A.employee = B.employee
```

- ✓ Aliases help with writing & reading the code more easily

# SYNTAX

Table A

Table B

```
SELECT * FROM TableA A  
INNER JOIN TableB B  
ON A.employee = B.employee
```

- ✓ Aliases help with writing & reading the code more easily

# SYNTAX

Table A

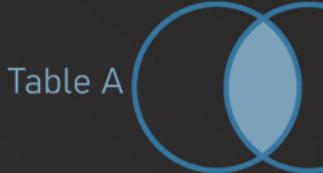


Table B

```
SELECT employee FROM TableA A  
INNER JOIN TableB B  
ON A.employee = B.employee
```

employee	city	sales
Sandra	Frankfurt	500
Sabine	Munich	300
Peter	Hamburg	200
Manuel	Hamburg	400
Michael	Munich	100
Frank	Frankfurt	100

employee	bonus
Sandra	YES
Sabine	YES
Peter	NO
Manuel	YES
Simon	NO

# SYNTAX

Table A

Table B

```
SELECT A.employee FROM TableA A  
INNER JOIN TableB B  
ON A.employee = B.employee
```

employee	city	sales
Sandra	Frankfurt	500
Sabine	Munich	300
Peter	Hamburg	200
Manuel	Hamburg	400
Michael	Munich	100
Frank	Frankfurt	100

employee	bonus
Sandra	YES
Sabine	YES
Peter	NO
Manuel	YES
Simon	NO

# SYNTAX

Table A

Table B

```
SELECT A.employee, sales FROM TableA A  
INNER JOIN TableB B  
ON A.employee = B.employee
```

employee	city	sales
Sandra	Frankfurt	500
Sabine	Munich	300
Peter	Hamburg	200
Manuel	Hamburg	400
Michael	Munich	100
Frank	Frankfurt	100

employee	bonus
Sandra	YES
Sabine	YES
Peter	NO
Manuel	YES
Simon	NO

# SYNTAX

employee	city	sales
Sandra	Frankfurt	500
Sabine	Munich	300
Peter	Hamburg	200
Manuel	Hamburg	400
Michael	Munich	100
Frank	Frankfurt	100

employee	bonus
Sandra	YES
Sabine	YES
Peter	NO
Manuel	YES
Simon	NO

Table A      Table B

✓ Always need a common column / reference

✓ INNER JOIN: Only rows where reference column value is in both tables

✓ Order of tables (A and B / B and A) does not matter

✓ Repeated values in either table will also be repeated

# Challenge

The airline company wants to understand in which category they sell most tickets.

How many people choose seats in the category

- Business
- Economy or
- Comfort?

You need to work on the seats table and the boarding\_passes table.

Result

	fare_conditions	count
	character varying (10)	bigint
1	Business	621092
2	Comfort	112098
3	Economy	2476771

# FULL OUTER JOIN

# FULL OUTER JOIN

Table A

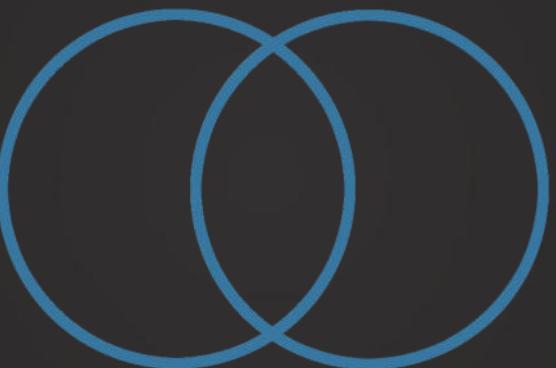


Table B

# FULL OUTER JOIN

Table A

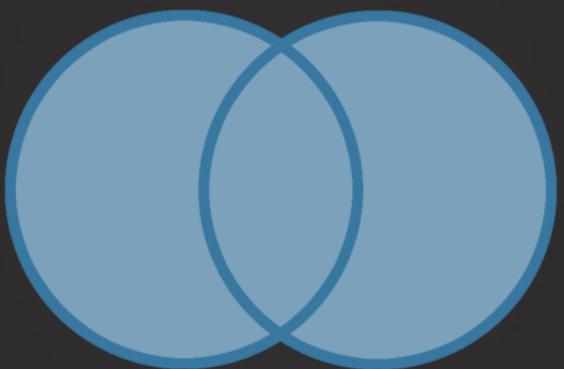


Table B

# FULL OUTER JOIN

**sales table**

employee	city	sales
Sandra	Frankfurt	500
Sabine	Munich	300
Peter	Hamburg	200
Manuel	Hamburg	400
Michael	Munich	100
Frank	Frankfurt	100

Do not appear in the bonus table

Table A



Table B

**bonus table**

employee	bonus
Sandra	YES
Sabine	YES
Peter	NO
Manuel	YES
Simon	NO

Do not appear in the sales table

# FULL OUTER JOIN

employee	city	sales
Sandra	Frankfurt	500
Sabine	Munich	300
Peter	Hamburg	200
Manuel	Hamburg	400
Michael	Munich	100
Frank	Frankfurt	100

employee	bonus
Sandra	YES
Sabine	YES
Peter	NO
Manuel	YES
Simon	NO



Table A

Table B

employee	city	sales	bonus
Sandra	Frankfurt	500	YES
Sabine	Munich	300	YES
Peter	Hamburg	200	NO
Manuel	Hamburg	400	YES

# FULL OUTER JOIN

employee	city	sales
Sandra	Frankfurt	500
Sabine	Munich	300
Peter	Hamburg	200
Manuel	Hamburg	400
Michael	Munich	100
Frank	Frankfurt	100

employee	bonus
Sandra	YES
Sabine	YES
Peter	NO
Manuel	YES
Simon	NO

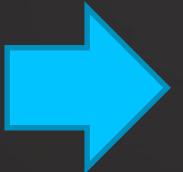


Table A

Table B

employee	city	sales	bonus
Sandra	Frankfurt	500	YES
Sabine	Munich	300	YES
Peter	Hamburg	200	NO
Manuel	Hamburg	400	YES
Michael	Munich	100	
Frank	Frankfurt	100	

# FULL OUTER JOIN

employee	city	sales
Sandra	Frankfurt	500
Sabine	Munich	300
Peter	Hamburg	200
Manuel	Hamburg	400
Michael	Munich	100
Frank	Frankfurt	100

employee	bonus
Sandra	YES
Sabine	YES
Peter	NO
Manuel	YES
Simon	NO



Table A

Table B

employee	city	sales	bonus
Sandra	Frankfurt	500	YES
Sabine	Munich	300	YES
Peter	Hamburg	200	NO
Manuel	Hamburg	400	YES
Michael	Munich	100	null
Frank	Frankfurt	100	null

# FULL OUTER JOIN

employee	city	sales
Sandra	Frankfurt	500
Sabine	Munich	300
Peter	Hamburg	200
Manuel	Hamburg	400
Michael	Munich	100
Frank	Frankfurt	100

employee	bonus
Sandra	YES
Sabine	YES
Peter	NO
Manuel	YES
Simon	NO



Table A

Table B

employee	city	sales	bonus
Sandra	Frankfurt	500	YES
Sabine	Munich	300	YES
Peter	Hamburg	200	NO
Manuel	Hamburg	400	YES
Michael	Munich	100	null
Frank	Frankfurt	100	null
	null	null	NO

# FULL OUTER JOIN

employee	city	sales
Sandra	Frankfurt	500
Sabine	Munich	300
Peter	Hamburg	200
Manuel	Hamburg	400
Michael	Munich	100
Frank	Frankfurt	100

employee	bonus
Sandra	YES
Sabine	YES
Peter	NO
Manuel	YES
Simon	NO



Table A



Table B

employee	city	sales	bonus
Sandra	Frankfurt	500	YES
Sabine	Munich	300	YES
Peter	Hamburg	200	NO
Manuel	Hamburg	400	YES
Michael	Munich	100	null
Frank	Frankfurt	100	null
null	null	null	NO

# FULL OUTER JOIN

employee	city	sales
Sandra	Frankfurt	500
Sabine	Munich	300
Peter	Hamburg	200
Manuel	Hamburg	400
Michael	Munich	100
Frank	Frankfurt	100

employee	bonus
Sandra	YES
Sabine	YES
Peter	NO
Manuel	YES
Simon	NO



Table A

Table B

bonus.employee	sales.employee	city	sales	bonus
Sandra	Sandra	Frankfurt	500	YES
Sabine	Sabine	Munich	300	YES
Peter	Peter	Hamburg	200	NO
Manuel	Manuel	Hamburg	400	YES
null	Michael	Munich	100	null
null	Frank	Frankfurt	100	null
Simon	null	null	null	NO

# FULL OUTER JOIN

employee	city	sales
Sandra	Frankfurt	500
Sabine	Munich	300
Peter	Hamburg	200
Manuel	Hamburg	400
Michael	Munich	100
Frank	Frankfurt	100

employee	bonus
Sandra	YES
Sabine	YES
Peter	NO
Manuel	YES
Simon	NO



Table A



Table B

bonus.employee	sales.employee	city	sales	bonus
null	Sandra	Frankfurt	500	YES
null	Sabine	Munich	300	YES
null	Peter	Hamburg	200	NO
null	Manuel	Hamburg	400	YES
null	Michael	Munich	100	null
null	Frank	Frankfurt	100	null
Simon	null	null	null	NO

# FULL OUTER JOIN

employee	city	sales
Sandra	Frankfurt	500
Sabine	Munich	300
Peter	Hamburg	200
Manuel	Hamburg	400
Michael	Munich	100
Frank	Frankfurt	100

employee	bonus
Sandra	YES
Sabine	YES
Peter	NO
Manuel	YES
Simon	NO

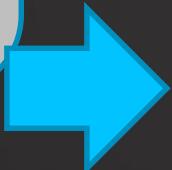


Table A

Table B

bonus.employee	sales.employee	city	sales	bonus
null	Sandra	Frankfurt	500	YES
null	Sabine	Munich	300	YES
null	Peter	Hamburg	200	NO
null	Manuel	Hamburg	400	YES
null	Michael	Munich	100	null
null	Frank	Frankfurt	100	null
Simon	null	null	null	NO

# SYNTAX

Table A



Table B



```
SELECT * FROM TableA  
FULL OUTER JOIN TableB  
ON TableA.employee = TableB.employee
```

# SYNTAX

Table A



Table B



```
SELECT * FROM TableB  
FULL OUTER JOIN TableA  
ON TableA.employee = TableB.employee
```

# SYNTAX

Table A



Table B

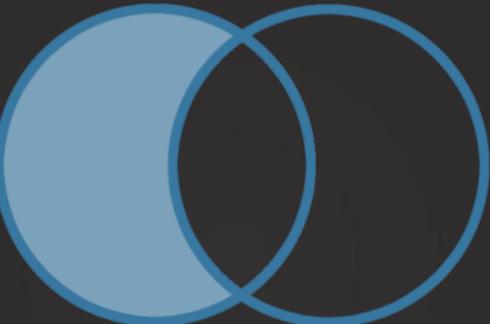


```
SELECT * FROM TableB  
FULL OUTER JOIN TableA  
ON TableA.employee = TableB.employee  
WHERE condition
```

# FULL OUTER JOIN

Table A

Table B



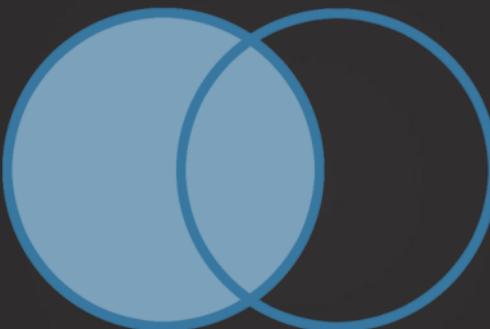
```
SELECT * FROM TableB  
FULL OUTER JOIN TableA  
ON TableA.employee = TableB.employee  
WHERE TableB.anycolumn IS null
```

# LEFT OUTER JOIN

# LEFT OUTER JOIN

Table A

Table B



# LEFT OUTER JOIN

**sales table**

employee	city	sales
Sandra	Frankfurt	500
Sabine	Munich	300
Peter	Hamburg	200
Manuel	Hamburg	400
Michael	Munich	100
Frank	Frankfurt	100

Do not appear in the bonus table

Table A



**bonus table**

employee	bonus
Sandra	YES
Sabine	YES
Peter	NO
Manuel	YES
Simon	NO

Do not appear in the sales table

# LEFT OUTER JOIN

employee	city	sales
Sandra	Frankfurt	500
Sabine	Munich	300
Peter	Hamburg	200
Manuel	Hamburg	400
Michael	Munich	100
Frank	Frankfurt	100

employee	bonus
Sandra	YES
Sabine	YES
Peter	NO
Manuel	YES
Simon	NO

LEFT



RIGHT

Table A

Table B

b.employee	s.employee	city	sales	bonus
Sandra	Sandra	Frankfurt	500	YES
Sabine	Sabine	Munich	300	YES
Peter	Peter	Hamburg	200	NO
Manuel	Manuel	Hamburg	400	YES
null	Michael	Munich	100	null
null	Frank	Frankfurt	100	null

# LEFT OUTER JOIN

employee	city	sales
Sandra	Frankfurt	500
Sabine	Munich	300
Peter	Hamburg	200
Manuel	Hamburg	400
Michael	Munich	100
Frank	Frankfurt	100

employee	bonus
Sandra	YES
Sabine	YES
Peter	NO
Manuel	YES
Simon	NO

LEFT



RIGHT

Table A

Table B

b.employee	s.employee	city	sales	bonus
Sandra	Sandra	Frankfurt	500	YES
Sabine	Sabine	Munich	300	YES
Peter	Peter	Hamburg	200	NO
Manuel	Manuel	Hamburg	400	YES
null	Michael	Munich	100	null
null	Frank	Frankfurt	100	null
Simon	null	null	null	NO

# SYNTAX

Table A



Table B

```
SELECT * FROM TableA  
LEFT OUTER JOIN TableB  
ON TableA.employee = TableB.employee
```

# Challenge

The flight company is trying to find out what their most popular seats are.

Try to find out which seat has been chosen most frequently.  
Make sure all seats are included even if they have never been booked.

Are there seats that have never been booked?

Result

	seat_no character varying (4)	count bigint
1	1A	53559
2	4A	53181
3	2A	53145

# Challenge

Try to find out which line (A, B, ..., H) has been chosen most frequently.

	seat_no	count
1	1A	53559
2	4A	53181
3	2A	53145

Result

1	A	751618
2	D	652188
3	C	596921

# Challenge

You want to create a tier list in the following way:

1. Rating is 'PG' or 'PG-13' or length is more than 210 min:  
'Great rating or long (tier 1)'
2. Description contains 'Drama' and length is more than 90min:  
'Long drama (tier 2)'
3. Description contains 'Drama' and length is not more than 90min:  
'Shcity drama (tier 3)'
4. Rental\_rate less than \$1:  
'Very cheap (tier 4)'

If one movie can be in multiple categories it gets the higher tier assigned.  
How can you filter to only those movies that appear in one of these 4 tiers?

Result

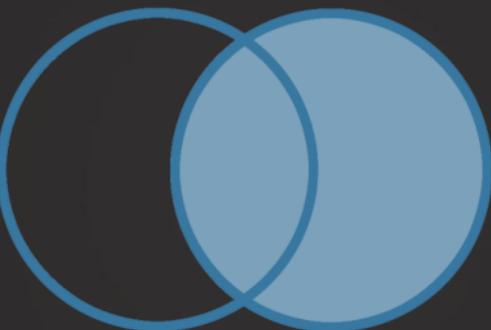
	title text	case text
1	ACADEMY DINOSAUR	Great rating or very long (tier 1)
2	AGENT TRUMAN	Great rating or very long (tier 1)
3	AIRPLANE SIERRA	Great rating or very long (tier 1)
4	ALABAMA DEVIL	Great rating or very long (tier 1)
5	ALAMO VIDEOTAPE	Very cheap (tier 4)

# RIGHT OUTER JOIN

# RIGHT OUTER JOIN

Table A

Table B



# FULL OUTER JOIN

Table A



Table B

**sales table**

employee	city	sales
Sandra	Frankfurt	500
Sabine	Munich	300
Peter	Hamburg	200
Manuel	Hamburg	400
Michael	Munich	100
Frank	Frankfurt	100

**bonus table**

employee	bonus
Sandra	YES
Sabine	YES
Peter	NO
Manuel	YES
Simon	NO

Do not appear in the bonus table

Do not appear in the sales table

# FULL OUTER JOIN

employee	city	sales
Sandra	Frankfurt	500
Sabine	Munich	300
Peter	Hamburg	200
Manuel	Hamburg	400
Michael	Munich	100
Frank	Frankfurt	100

employee	bonus
Sandra	YES
Sabine	YES
Peter	NO
Manuel	YES
Simon	NO

LEFT



RIGHT

Table A

Table B

b.employee	s.employee	city	sales	bonus
Sandra	Sandra	Frankfurt	500	YES
Sabine	Sabine	Munich	300	YES
Peter	Peter	Hamburg	200	NO
Manuel	Manuel	Hamburg	400	YES
Simon	null	null	null	NO

	Michael	Munich	100	
	Frank	Frankfurt	100	

# SYNTAX

Table A



Table B



```
SELECT * FROM TableA  
RIGHT OUTER JOIN TableB  
ON TableA.employee = TableB.employee
```

# SYNTAX

Table A



Table B

```
SELECT * FROM TableB  
LEFT OUTER JOIN TableA  
ON TableA.employee = TableB.employee
```

# Challenge

The company wants to run a phone call campaing on all customers in Texas (=district).

What are the customers (first\_name, last\_name, phone number and their district) from Texas?

Are there any (old) addresses that are not related to any customer?

## Result

first_name text	last_name text	phone text	district text
JENNIFER	DAVIS	860452626434	Texas
KIM	CRUZ	909029256431	Texas
RICHARD	MCCRARY	262088367001	Texas
BRYAN	HARDISON	775235029633	Texas
IAN	STILL	239357986667	Texas

address_id integer	address text
1	47 MySakila Drive
2	28 MySQL Boulevard
3	23 Workhaven Lane
4	1411 Lillydale Drive

# Multiple join conditions

# Multiple join conditions

**sales table**

first_name	last_name	city	sales
Sandra	Jones	Frankfurt	500
Sandra	Williams	Munich	300
Peter	Davis	Hamburg	200
Manuel	White	Hamburg	400
Michael	Anderson	Munich	100
Frank	Wilson	Frankfurt	100

**bonus table**

first_name	last_name	bonus
Sandra	Jones	YES
Sandra	Williams	YES
Peter	Davis	NO
Manuel	White	YES
Simon	Taylor	NO

# SYNTAX

```
SELECT * FROM TableA a  
INNER JOIN TableB b  
ON a.first_name = b.first_name  
AND a.last_name = b.last_name
```

## Expert tip

```
SELECT * FROM TableA a  
INNER JOIN TableB b  
ON a.first_name = b.first_name  
AND a.last_name = 'Jones'
```

# Expert tip

```
SELECT * FROM TableA a  
INNER JOIN TableB b  
ON a.first_name = b.first_name  
AND a.last_name = 'Jones'
```

More performance-efficient!

```
SELECT * FROM TableA a  
INNER JOIN TableB b  
ON a.first_name = b.first_name  
WHERE a.last_name = 'Jones'
```

# Challenge

The company wants to run a phone call campaing on all customers in Texas (=district).

What are the customers (first\_name, last\_name, phone number and their district) from Texas?

Are there any (old) addresses that are not related to any customer?

## Result

first_name text	last_name text	phone text	district text
JENNIFER	DAVIS	860452626434	Texas
KIM	CRUZ	909029256431	Texas
RICHARD	MCCRARY	262088367001	Texas
BRYAN	HARDISON	775235029633	Texas
IAN	STILL	239357986667	Texas

address_id integer	address text
1	47 MySakila Drive
2	28 MySQL Boulevard
3	23 Workhaven Lane
4	1411 Lillydale Drive

# Two join tables

**sales table**

first_name	last_name	city	sales
Sandra	Jones	Frankfurt	500
Sandra	Williams	Munich	300
Peter	Davis	Hamburg	200
Manuel	White	Hamburg	400
Michael	Anderson	Munich	100
Frank	Wilson	Frankfurt	100

**bonus table**

first_name	last_name	bonus
Sandra	Jones	YES
Sandra	Williams	YES
Peter	Davis	NO
Manuel	White	YES
Simon	Taylor	NO

# Joining multiple tables

**sales table**

employee	city_id	sales
Sandra	1	500
Sabine	2	300
Peter	3	200
Manuel	3	400
Michael	3	100
Frank	1	100

**city table**

city_id	city	country_id
1	Frankfurt	1
2	Munich	1
3	New York	2

**country table**

country_id	country
1	Germany
2	USA

# Joining multiple tables

**sales table**

employee	city_id	sales
Sandra	1	500
Sabine	2	300
Peter	3	200
Manuel	3	400
Michael	3	100
Frank	1	100

**city table**

city_id	city	country_id
1	Frankfurt	1
2	Munich	1
3	New York	2

**country table**

country_id	country
1	Germany
2	USA

employee	country
Sandra	Germany
Sabine	Germany
Peter	USA
Manuel	USA
Michael	USA
Frank	Germany

# Joining multiple tables

**sales table**

employee	city_id	sales
Sandra	1	500
Sabine	2	300
Peter	3	200
Manuel	3	400
Michael	3	100
Frank	1	100

**city table**

city_id	city	country_id
1	Frankfurt	1
2	Munich	1
3	New York	2

**country table**

country_id	country
1	Germany
2	USA

employee	country
Sandra	Germany
Sabine	Germany
Peter	USA
Manuel	USA
Michael	USA
Frank	Germany

# Joining multiple tables

**sales table**

employee	city_id	sales
Sandra	1	500
Sabine	2	300
Peter	3	200
Manuel	3	400
Michael	3	100
Frank	1	100

**city table**

city_id	city	country_id
1	Frankfurt	1
2	Munich	1
3	New York	2

**country table**

country_id	country
1	Germany
2	USA

employee	country_id
Sandra	1
Sabine	1
Peter	2
Manuel	2
Michael	2
Frank	1

employee	country
Sandra	Germany
Sabine	Germany
Peter	USA
Manuel	USA
Michael	USA
Frank	Germany

# SYNTAX

```
SELECT employee, ci.country_id FROM sales s  
INNER JOIN city ci  
ON s.city_id = ci.city_id
```

employee	country_id
Sandra	1
Sabine	1
Peter	2
Manuel	2
Michael	2
Frank	1

# SYNTAX

```
SELECT employee, co.country FROM sales s  
INNER JOIN city ci  
ON s.city_id = ci.city_id  
INNER JOIN country co  
ON ci.country_id = co.country_id
```

employee	country
Sandra	Germany
Sabine	Germany
Peter	USA
Manuel	USA
Michael	USA
Frank	Germany

# SYNTAX

```
SELECT employee, co.country FROM sales s  
INNER JOIN country co  
ON ci.country_id = co.country_id  
INNER JOIN city ci  
ON s.city_id = ci.city_id
```

INNER JOIN:

Table order doesn't matter!

employee	country
Sandra	Germany
Sabine	Germany
Peter	USA
Manuel	USA
Michael	USA
Frank	Germany

# Joining multiple tables

**sales table**

employee	city_id	sales
Sandra	1	500
Sabine	2	300
Peter	3	200
Manuel	3	400
Michael	3	100
Frank	4	100

**city table**

city_id	city	country_id
1	Frankfurt	1
2	Munich	1
3	New York	2

**country table**

country_id	country
1	Germany
2	USA

employee	country
Sandra	Germany
Sabine	Germany
Peter	USA
Manuel	USA
Michael	USA
Frank	null

# SYNTAX

```
SELECT employee, co.country FROM sales s  
LEFT JOIN city ci  
ON s.city_id = ci.city_id  
LEFT JOIN country co  
ON ci.country_id = co.country_id
```

employee	country
Sandra	Germany
Sabine	Germany
Peter	USA
Manuel	USA
Michael	USA
Frank	null

# SYNTAX

```
SELECT employee, co.country FROM city ci  
LEFT JOIN sales s  
ON s.city_id = ci.city_id  
LEFT JOIN country co  
ON ci.country_id = co.country_id
```

employee	country
Sandra	Germany
Sabine	Germany
Peter	USA
Manuel	USA
Michael	USA
Frank	null

# Challenge

The company wants customize their campaigns to customers depending on the country they are from.

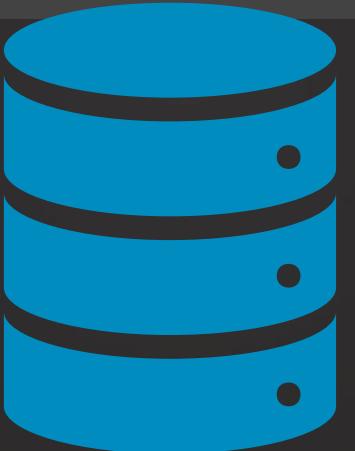
Which customers are from Brazil?

Write a query to get first\_name, last\_name, email and the country from all customers from Brazil.

## Result

first_name text	last_name text	email text	country text
CLAYTON	BARBEE	CLAYTON.BARBEE@sakilacustomer.org	Brazil
JOSEPH	JOY	JOSEPH.JOY@sakilacustomer.org	Brazil
TAMARA	NGUYEN	TAMARA.NGUYEN@sakilacustomer.org	Brazil

# Day 7





UNION

# FULL OUTER JOIN

employee	city	sales
Sandra	Frankfurt	500
Sabine	Munich	300
Peter	Hamburg	200
Manuel	Hamburg	400
Michael	Munich	100
Frank	Frankfurt	100

employee	bonus
Sandra	YES
Sabine	YES
Peter	NO
Manuel	YES
Simon	NO

Combining columns

bonus.employee	sales.employee	city	sales	bonus
null	Sandra	Frankfurt	500	YES
null	Sabine	Munich	300	YES
null	Peter	Hamburg	200	NO
null	Manuel	Hamburg	400	YES
null	Michael	Munich	100	null
		Frankfurt	100	null
		null	null	NO

Table A

Table B

# UNION

New York

name	sales
Sandra	500
Maya	300
Peter	200

Delhi

name	sales
Sunita	600
Anil	400
Shanti	100

Combining multiple  
select statements

# UNION

New York

name	sales
Sandra	500
Maya	300
Peter	200

Delhi

name	sales
Sunita	600
Anil	400
Shanti	100

name	sales
Sunita	600
Anil	400
Shanti	100
Sandra	500
Maya	300
Peter	200

# SYNTAX

```
SELECT first_name, sales FROM vancouver  
UNION  
SELECT first_name, sales FROM delhi
```

3 Things to remember!

How columns are  
matched?

# 1st thing to remember

New York

name	sales
Sandra	500
Maya	300
Peter	200

Delhi

name	sales
Sunita	600
Anil	400
Shanti	100

Columns are matched  
by the order!

# 1st thing to remember

New York

name	sales
Sandra	500
Maya	300
Peter	200

Delhi

<i>first_name</i>	sales
Sunita	600
Anil	400
Shanti	100

# UNION

New York

name	sales
Sandra	500
Maya	300
Peter	200

Delhi

first_name	sales
Sunita	600
Anil	400
Shanti	100



name	sales
Sandra	500
Maya	300
Peter	200
Sunita	600
Anil	400
Shanti	100

# UNION

New York

name	sales
Sandra	500
Maya	300
Peter	200

Delhi

first_name	sales
Sunita	600
Anil	400
Shanti	100



name	sales
Sandra	500
Maya	300
Peter	200
Sunita	600
Anil	400
Shanti	100

# SYNTAX

```
SELECT first_name, sales FROM delhi  
UNION  
SELECT name, sales FROM vancouver
```

# UNION

New York

<b>name</b>	<b>sales</b>
Sandra	500
Maya	300
Peter	200

Delhi

<b>first_name</b>	<b>sales</b>
Sunita	600
Anil	400
Shanti	100

<b>first_name</b>	<b>sales</b>
Sunita	600
Anil	400
Shanti	100
Sandra	500
Maya	300
Peter	200

# SYNTAX

```
SELECT first_name, sales FROM delhi  
UNION  
SELECT first_name, sales FROM vancouver
```

Must be aware of the  
order in the match

# SYNTAX

```
SELECT first_name, sales FROM delhi  
UNION  
SELECT sales, first_name FROM vancouver
```

# UNION

New York

name	sales
Sandra	500
Maya	300
Peter	200

Delhi

first_name	sales
Sunita	600
Anil	400
Shanti	100

first_name	sales
Sunita	600
Anil	400
Shanti	100
500	Sandra
300	Maya
200	Peter

Data type  
must match!

# UNION

New York

name	sales
Sandra	500
Maya	300
Peter	200

Delhi

first_name	sales
Sunita	600
Anil	400
Shanti	100

No. of columns  
must match!



Data type  
must match!

# UNION

New York

name	sales
Sunita	600
Maya	300
Peter	200

Delhi

first_name	sales
Sunita	600
Anil	400
Shanti	100



first_name	sales
Sunita	600
Anil	400
Shanti	100
Maya	300
Peter	200

Duplicates are  
decoupled!

# UNION

New York

name	sales
Sunita	600
Maya	300
Peter	200



Delhi

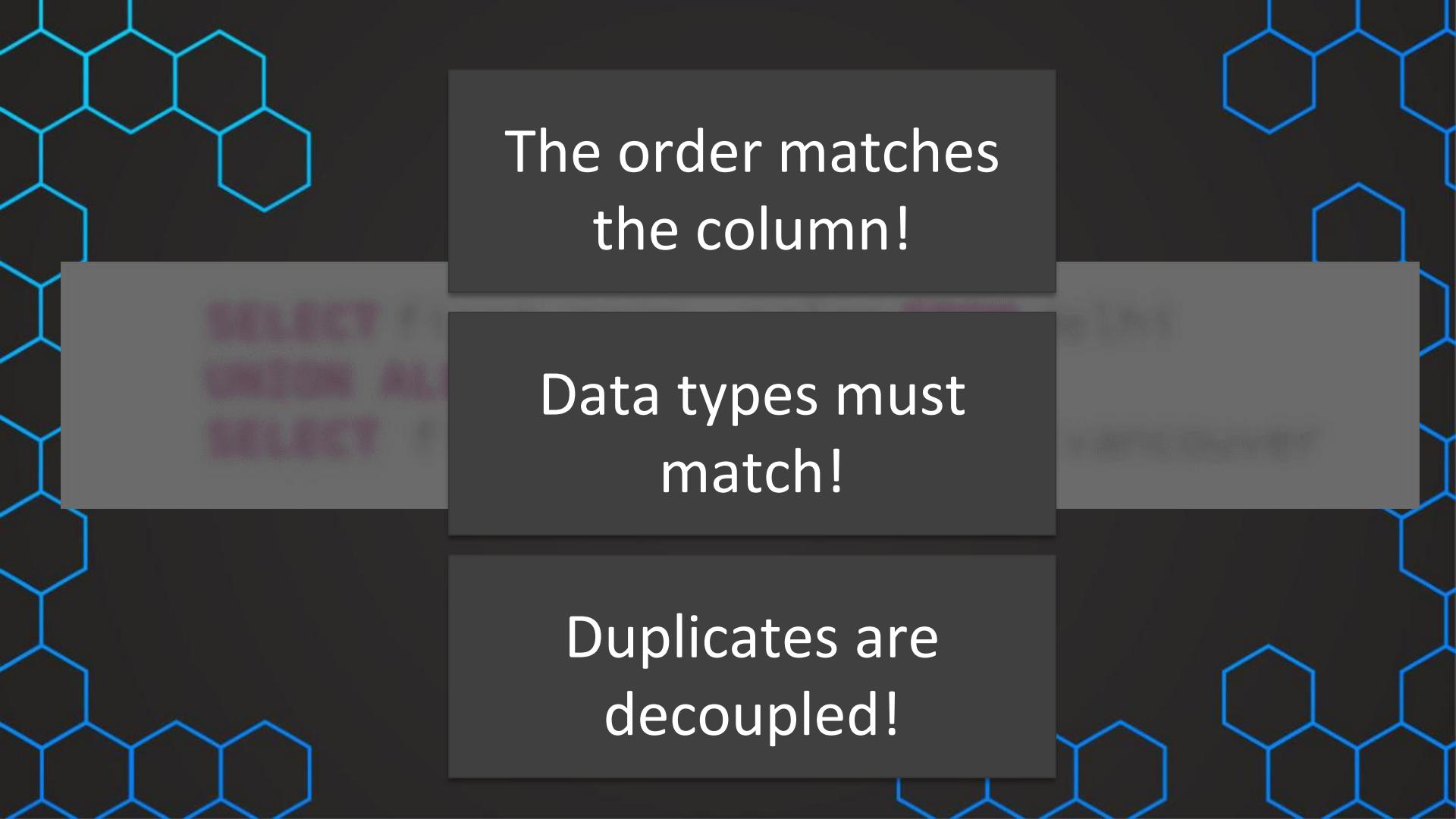
first_name	sales
Sunita	600
Anil	400
Shanti	100

UNION ALL

first_name	sales
Sunita	600
Anil	400
Shanti	100
Sunita	600
Maya	300
Peter	200

# SYNTAX

```
SELECT first_name, sales FROM delhi  
UNION ALL  
SELECT first_name, sales FROM vancouver
```



The order matches  
the column!

Data types must  
match!

Duplicates are  
decoupled!

# Correlated subqueries

name	sales
Sunita	600
Anil	400
Shanti	100
Sunita	300
Maya	300
Peter	200
Max	100
Anna	400



name	sales
Sunita	600
Anil	400
Anna	400

Get all people that are above average!

```
SELECT first_name, sales FROM employees  
WHERE sales >  
(SELECT AVG(sales) FROM employees)
```

300

# Correlated subqueries

name	sales	city
Sunita	600	Delhi
Anil	400	Delhi
Shanti	100	Delhi
Sunita	300	Dallas
Maya	300	Dallas
Peter	200	Dallas
Max	100	Berlin
Anna	400	Berlin

Get all people that are above  
average of their city!

Correlated subquery!

# Correlated subqueries

name	sales	city
Sunita	600	Delhi
Anil	400	Delhi
Shanti	100	Delhi
Sunita	300	Dallas
Maya	300	Dallas
Peter	200	Dallas
Max	100	Berlin
Anna	400	Berlin

Get all people that are above  
average of their city!

```
SELECT first_name, sales FROM employees
WHERE sales >
    (... correlated subquery ...)
```

# Correlated subqueries

name	sales	city
Sunita	600	Delhi
Anil	400	Delhi
Shanti	100	Delhi
Sunita	300	Dallas
Maya	300	Dallas
Peter	200	Dallas
Max	100	Berlin
Anna	400	Berlin

Get all people that are above  
average of their city!

```
SELECT first_name, sales FROM employees
WHERE sales >
    (SELECT AVG(sales) FROM employees
     ... )
```

# Correlated subqueries

name	sales	city
Sunita	600	Delhi
Anil	400	Delhi
Shanti	100	Delhi
Sunita	300	Dallas
Maya	300	Dallas
Peter	200	Dallas
Max	100	Berlin
Anna	400	Berlin

Get all people that are above  
average of their city!

```
SELECT first_name, sales FROM employees e1
WHERE sales >
    (SELECT AVG(sales) FROM employees e2
     WHERE e1.city=e2.city )
```

Evaluated for every single row!

# Correlated subqueries

name	sales	city
Sunita	600	Delhi
Anil	400	Delhi
Shanti	100	Delhi
Sunita	300	Dallas
Maya	300	Dallas
Peter	200	Dallas
Max	100	Berlin
Anna	400	Berlin

```
WHERE sales >  
(SELECT AVG(sales) FROM employees e2  
WHERE e1.city=e2.city )
```

~366.67

```
SELECT first_name, sales FROM employees e1  
WHERE sales >  
(SELECT AVG(sales) FROM employees e2  
WHERE e1.city=e2.city )
```

# Correlated subqueries

name	sales	city
Sunita	600	Delhi
Anil	400	Delhi
Shanti	100	Delhi
Sunita	300	Dallas
Maya	300	Dallas
Peter	200	Dallas
Max	100	Berlin
Anna	400	Berlin

```
WHERE sales >  
(SELECT AVG(sales) FROM employees e2  
WHERE e1.city=e2.city )
```

~366.67

```
SELECT first_name, sales FROM employees e1  
WHERE sales >  
(SELECT AVG(sales) FROM employees e2  
WHERE e1.city=e2.city )
```

# Correlated subqueries

name	sales	city
Sunita	600	Delhi
Anil	400	Delhi
Shanti	100	Delhi
Sunita	300	Dallas
Maya	300	Dallas
Peter	200	Dallas
Max	100	Berlin
Anna	400	Berlin

```
WHERE sales >  
(SELECT AVG(sales) FROM employees e2  
WHERE e1.city=e2.city )
```

~366.67

```
SELECT first_name, sales FROM employees e1  
WHERE sales >  
(SELECT AVG(sales) FROM employees e2  
WHERE e1.city=e2.city )
```

# Correlated subqueries

name	sales	city
Sunita	600	Delhi
Anil	400	Delhi
Shanti	100	Delhi
Sunita	300	Dallas
Maya	300	Dallas
Peter	200	Dallas
Max	100	Berlin
Anna	400	Berlin

```
WHERE sales >  
(SELECT AVG(sales) FROM employees e2  
WHERE e1.city=e2.city )
```

~366.67

```
SELECT first_name, sales FROM employees e1  
WHERE sales >  
(SELECT AVG(sales) FROM employees e2  
WHERE e1.city=e2.city )
```

# Correlated subqueries

name	sales	city
Sunita	600	Delhi
Anil	400	Delhi
Shanti	100	Delhi
Sunita	300	Dallas
Maya	300	Dallas
Peter	200	Dallas
Max	100	Berlin
Anna	400	Berlin

```
WHERE sales >  
(SELECT AVG(sales) FROM employees e2  
WHERE e1.city=e2.city )
```

~266.67

```
SELECT first_name, sales FROM employees e1  
WHERE sales >  
(SELECT AVG(sales) FROM employees e2  
WHERE e1.city=e2.city )
```

# Correlated subqueries

name	sales	city
Sunita	600	Delhi
Anil	400	Delhi
Shanti	100	Delhi
Sunita	300	Dallas
Maya	300	Dallas
Peter	200	Dallas
Max	100	Berlin
Anna	400	Berlin

```
WHERE sales >  
(SELECT AVG(sales) FROM employees e2  
WHERE e1.city=e2.city )
```

~266.67

```
SELECT first_name, sales FROM employees e1  
WHERE sales >  
(SELECT AVG(sales) FROM employees e2  
WHERE e1.city=e2.city )
```

# Correlated subqueries

name	sales	city
Sunita	600	Delhi
Anil	400	Delhi
Shanti	100	Delhi
Sunita	300	Dallas
Maya	300	Dallas
Peter	200	Dallas
Max	100	Berlin
Anna	400	Berlin

```
WHERE sales >  
(SELECT AVG(sales) FROM employees e2  
WHERE e1.city=e2.city )
```

250

```
SELECT first_name, sales FROM employees e1  
WHERE sales >  
(SELECT AVG(sales) FROM employees e2  
WHERE e1.city=e2.city )
```

# Correlated subqueries

name	sales	city
Sunita	600	Delhi
Anil	400	Delhi
Shanti	100	Delhi
Sunita	300	Dallas
Maya	300	Dallas
Peter	200	Dallas
Max	100	Berlin
Anna	400	Berlin

```
WHERE sales >  
(SELECT AVG(sales) FROM employees e2  
WHERE e1.city=e2.city )
```

250

```
SELECT first_name, sales FROM employees e1  
WHERE sales >  
(SELECT AVG(sales) FROM employees e2  
WHERE e1.city=e2.city )
```

# Correlated subqueries

name	sales	city
Sunita	600	Delhi
Anil	400	Delhi
Shanti	100	Delhi
Sunita	300	Dallas
Maya	300	Dallas
Peter	200	Dallas
Max	100	Berlin
Anna	400	Berlin

```
WHERE sales >  
  (SELECT AVG(sales) FROM employees e2  
   WHERE e1.city=e2.city )
```

Subquery gets evaluated  
for every single row!

```
SELECT first_name, sales FROM employees e1  
WHERE sales >  
  (SELECT AVG(sales) FROM employees e2  
   WHERE e1.city=e2.city )
```

# Correlated subqueries

name	sales	city
Sunita	600	Delhi
Anil	400	Delhi

```
WHERE sales >  
  (SELECT AVG(sales) FROM employees e2  
   WHERE e1.city=e2.city )
```

Subquery does not work independently!

first_name	last_name	city
Anna	400	Berlin

Subquery gets evaluated for every single row!

```
SELECT first_name, sales FROM employees e1  
WHERE sales >  
  (SELECT AVG(sales) FROM employees e2  
   WHERE e1.city=e2.city )
```

# Challenge

Show only those movie titles, their associated film\_id and replacement\_cost with the lowest replacement\_costs for in each rating category – also show the rating.

## Result

Data Output		Explain	Messages	Notifications
#	title text	film_id [PK] integer	replacement_cost numeric (5,2)	rating mpaa_rating
1	ANACONDA CONFESSIONS	23	9.99	R
2	CIDER DESIRE	150	9.99	PG
3	CONTROL ANTHEM	182	9.99	G

# Challenge

Show only those movie titles, their associated film\_id and the length that have the highest length in each rating category – also show the rating.

## Result

	title text	film_id [PK] integer	rating mpaa_rating	length smallint
1	CHICAGO NORTH	141	PG-13	185
2	CONTROL ANTHEM	182	G	185
3	CRYSTAL BREAKING	198	NC-17	184

# Correlated subqueries

name	sales	city	min
Sunita	600	Delhi	100
Anil	400	Delhi	100
Shanti	100	Delhi	100
Sunita	300	Dallas	200
Maya	300	Dallas	200
Peter	200	Dallas	200
Max	100	Berlin	100
Anna	400	Berlin	100

```
SELECT first_name, sales FROM employees e1  
WHERE sales >  
(SELECT AVG(sales) FROM employees e2  
WHERE e1.city=e2.city )
```

# Correlated subqueries

name	sales	city	min
Sunita	600	Delhi	100
Anil	400	Delhi	100
Shanti	100	Delhi	100
Sunita	300	Dallas	200
Maya	300	Dallas	200
Peter	200	Dallas	200
Max	100	Berlin	100
Anna	400	Berlin	100

```
SELECT first_name, sales,  
       (SELECT MIN(sales) FROM employees e3  
        WHERE e1.city=e3.city )  
FROM employees e1  
WHERE sales >  
      (SELECT AVG(sales) FROM employees e2  
       WHERE e1.city=e2.city )
```

# Challenge

Show all the payments plus the total amount for every customer as well as the number of payments of each customer.

## Result

	payment_id	customer_id	staff_id	amount	sum_amount	count_payments
	integer	smallint	smallint	numeric (5,2)	numeric	bigrnt
1	18497	1	2	9.99	118.68	32
2	28997	1	1	7.99	118.68	32
3	28993	1	2	5.99	118.68	32
4	28994	1	1	5.99	118.68	32

# Challenge

Show only those films with the highest replacement costs in their rating category plus show the average replacement cost in their rating category.

## Result

	title text	replacement_cost numeric (5,2)	rating mpaa_rating	avg numeric
1	ARABIA DOGMA	29.99	NC-17	20.1376190476190476
2	BALLROOM MOCKINGBIRD	29.99	G	20.1248314606741573
3	BLINDNESS GUN	29.99	PG-13	20.4025560538116592

# Challenge

Show only those payments with the highest payment for each customer's first name - including the payment\_id of that payment.

How would you solve it if you would not need to see the payment\_id?

## Result

	first_name	amount	payment_id
	text	numeric (5,2)	integer
1	MARY	9.99	18497
2	PATRICIA	10.99	29014
3	LINDA	10.99	29022

# Day 8



# Mid-course project



# Mid-course project



Junior data analyst



Senior data analyst



# Mid-course project



10 out of 14  
challenges



Senior data analyst



## Challenges

### Question 1:

### Level: Simple

## Topic: DISTINCT

Task: Create a list of all the different replacement costs of the films.

Question: Whats the lowest replacement cost?

**Answer:** 9.99

pgAdmin 4

File | Object | Tools | Help |

Browser

Databases (1) + Servers (0)

- o PostgreSQL 11.5
- o PostgreSQL 10.13
- o PostgreSQL 14

Query Editor Query History

```
1 SELECT * FROM aircrafts;
```

aircraft_code	model	range	max_fuel
1	772	Boeing 777-300	11100
2	763	Boeing 767-300	7900
3	809	Airbus A380-800	10000
4	808	Airbus A380-800	9700
5	221	Airbus A321-100	3400
6	219	Airbus A321-100	4700
7	773	Beech 777-300	4000
8	201	Cessna 208 Caravan	1200
9	200	Cessna 172	1200

## Solutions

Here are the solutions. Be aware that there can be multiple correct solutions and ways of solving the challenges.

If your solution is different this doesn't mean that your solution is not correct.

You should just find the correct answers.

### Solution 1:

```
1 | SELECT DISTINCT replacement_cost  
2 | FROM film  
3 | ORDER BY 1
```

# Mid-course project

## Additional notes

Do the questions in one or two days

Sometimes multiple solutions are possible

Evaluate yourself



# Mid-course project



10 out of 14  
challenges



Senior data analyst

# Evaluation

## Questions right

Less than 6  
questions

Between 6 and 8  
questions

Between 9 and 11  
questions

At least 12  
questions

## Skill level

Basic skills;  
Recommendation: Work through lectures again

Decent skills;  
Recommendation: Work the solutions again

Good skills;  
Congrats on what you have achieved!

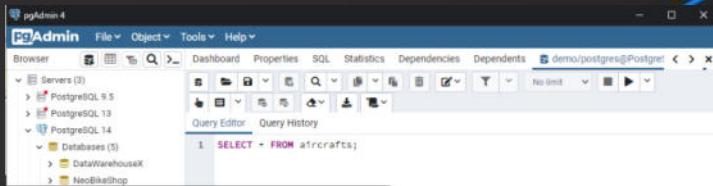
Very good skills;  
Outstanding!

# The Project



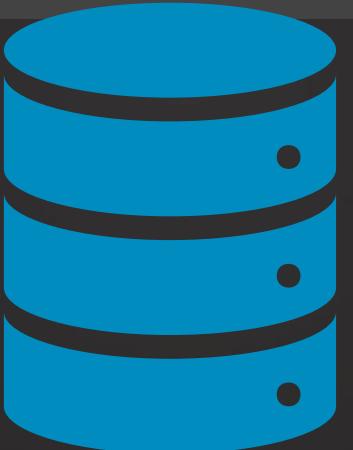
Data that is very  
important for the success!

YOU:  
Data Analyst



8	CN1	Cessna 208 Caravan	1200
9	CR1	Bombardier CRJ-200	2700

# Day 9



# Managing tables

CREATE

Database

Table

ALTER

Schemas

DROP

[Data Definition]

# Managing tables

CREATE

INSERT

ALTER

Data Definition

UPDATE

Data Manipulation

DROP

DELETE

Data Structures

Data itself

# Managing tables

CREATE

INSERT

Data Types

ALTER

Data Definition

UPDATE

Data Manipulation

Constraints

DROP

DELETE

Primary & Foreign Keys

Data Structures

Data itself

Views

# Creating database

Very simple

```
CREATE DATABASE <database_name>;
```

# Dropping database

Very simple

```
DROP DATABASE <database_name>;
```

Be very careful with dropping database objects!

# Data Types

Important when creating tables

Understanding data types

Storage size

Differences

When to use  
which one?

Possible operations

How to store ZIP codes?

# Data Types

Numeric

Strings

Date/Time

Other

<https://www.postgresql.org/docs/current/datatype.html>

# Data Types

## Numeric

Type	Storage size	Range	Notes
INT	4 bytes	-2147483648 to +2147483647	Typical choice
SMALLINT	2 bytes	-32768 to +32767	Small integers
BIGINT	8 bytes	-9223372036854775808 to +9223372036854775807	Large integers
DECIMAL	variable	up to 131072 digits before the decimal point; up to 16383 digits after the decimal point	user-defined precision
SERIAL	variable	1 to 2147483647	autoincrementing integer

# Data Types

## Numeric

Type	Storage size	Range	Notes
INT	4 bytes	-2147483648 to +2147483647	Typical choice
SMALLINT	2 bytes	-32768 to +32767	Small integers
BIGINT	8 bytes		
NUMERIC	variable		

`numeric(precision, scale)`

Precision: total count of digits

Scale: count of decimal places

24.99    2 decimal places

`numeric(4,2)`

# Data Types

## Strings

Type	Storage size	Example	Notes
character varying(n), varchar(n)	variable-length with limit	Any text, "Hello"	Less flexible to change!
character(n), char(n)	fixed-length, blank padded	"M" or "F"	Not better in performance!
text	variable unlimited length	Any text, "Hello"	Winner!

Which one to choose?

# Data Types

Strings

How about ZIP codes or phones numbers?

ZIP code: 0142

phone: 0049-234422

They don't have a numerical meaning!

Rather stored as string!

# Data Types

## Date/time

Type	Description	Example
date	Just date without time	'2022-11-28'
time (with/without time zone)	Just time without date	'01:02:03.678'
timestamp (with/without time zone)	Date and time	'2022-11-28 01:02:03.678+02'
intervals	Time interval	'3 days 01:02:03.678'

# Data Types

## Others

Type	Description	Example	Range
boolean	<b>state of true or false</b>	is_in_stock	TRUE, FALSE, NULL
		Allowed input:	
		true	false
		yes	no
		1	0
		on	off

# Data Types

## Others

Type	Description	Example	Range
boolean	<b>state of true or false</b>	is_in_stock	TRUE, FALSE, NULL
enum	<b>A value of a list of ordered values</b>	movie_rating	User-defined

```
CREATE TYPE mppa_rating AS ENUM ('G', 'PG', [...])
```

# Data Types

## Others

Type	Description	Example	Range
bool	<b>name</b> <b>phone</b>		NULL
enum	Peter	{ '+48-4893245123' , '+46-323245143' }	
array	Frank	{ '+41-39190643' }	type
	Maya	{ '+42-66764453' , '+434567651234' , '+43123676514' }	

```
SELECT name, phone FROM customers
```

# Data Types

## Others

Type	Description		Example	Range
bool	<b>name</b>	<b>phone</b>		NULL
enum	Peter	+48-4893245123		
array	Frank	+41-39190643		type
	Maya	+42-66764453		

```
SELECT name, phone[1] FROM customers
```

# Data Types

## Others

Type	Description	Example	Range
bool	<b>name</b> <b>phone</b>		NULL
enum	Maya	{ '+42-66764453', '+434567651234', '+43123676514' }	
array	Stores a list of values	text[] or int[]	Depending on type

```
SELECT name, phone[1] FROM customers  
WHERE '+42-66764453' = ANY (phones)
```

# Constraints

Column name

Data type

Constraints

# Constraints

## What is a constraint?

Defined when table is created

Used to define rules for the data in a table

Prevent insert of invalid data

Can be on column or table level

# Constraints

## COLUMN CONSTRAINTS

What constraints do we have?

NOT NULL

Ensures that a column cannot have a NULL value

UNIQUE

Ensures that all values in a column are different

DEFAULT

Sets a default value for a column if no value is specified

```
ERROR: insert or update on table "director" violates foreign key constraint "director_address_id_fkey"
DETAIL: Key (address_id)=(0) is not present in table "address".
SQL state: 23503
```

REFERENCES

Ensures referential integrity (only values of another column can be used)

CHECK

Ensures that the values in a column satisfies a specific condition

# Constraints

TABLE  
CONSTRAINTS

What constraints do we have?

PRIMARY KEY ( column [ ... ] )

UNIQUE ( column [ ... ] )

CHECK ( search\_condition )

# Primary & Foreign Key

## PRIMARY KEY

One or multiple columns that uniquely identify each row in a table

UNIQUE

NOT NULL

	film_id	title	description
	[PK] integer	text	text
1	1	ACADEMY DINOSAUR	A Epic Drama of a
2	2	ACE GOLDFINGER	A Astounding Epis
3	3	ADAPTATION HOLES	A Astounding Refl

# Primary & Foreign Key

## FOREIGN KEY

A Column (or multiple) that refers to the primary in another table

REFERENCING table  
CHILD table

	payment_id	customer_id	staff_id	
	integer	smallint	smallint	
1	16050	269	2	
2	16051	269	1	
3	16052	269	2	

Data Output Explain Messages Notifications

	customer_id [PK] integer	store_id smallint	first_name text		last_name text
1		1	1	MARY	SMITH
2		2	1	PATRICIA	JOHNSON
3		3	1	LINDA	

REFERENCED table  
PARENT table

# Primary & Foreign Key

## Notes

1. Foreign key does not need to be unique
2. Primary key and foreign keys are usually the columns to join tables
3. Can be created also in table creation process

Data Output Explain Messages Notifications

	payment_id integer	customer_id smallint	staff_id smallint	rental_id integer	amount numeric (5,2)	payment_date timestamp with time zone	lock
1	16050	269	2	7	1.99	2020-01-24 22:40:19.996577+01	
2	16051	269	1	98	0.99	2020-01-25 16:16:50.996577+01	
3	16052	269	2	678	6.99	2020-01-28 22:44:14.996577+01	

# CREATE TABLE

```
CREATE TABLE <table_name>
```

# CREATE TABLE

```
CREATE TABLE <table_name> (  
    column_name1 TYPE  
)
```

# CREATE TABLE

```
CREATE TABLE <table_name> (
    column_name1 TYPE,
    column_name2 TYPE
)
```

# CREATE TABLE

```
CREATE TABLE staff(  
column_name1 TYPE,  
column_name2 TYPE  
)
```

# CREATE TABLE

```
CREATE TABLE staff(  
staff_id INT,  
column_name2 TYPE  
)
```

# CREATE TABLE

```
CREATE TABLE staff(  
staff_id INT,  
name VARCHAR(50)  
)
```

# CREATE TABLE

```
CREATE TABLE staff(  
staff_id INT,  
name VARCHAR(50)  
)
```

# CREATE TABLE

```
CREATE TABLE <table_name>(  
    column_name1 TYPE [CONSTRAINT] ,  
    column_name2 TYPE [CONSTRAINT] ,  
    [...] )
```

# CREATE TABLE

```
CREATE TABLE staff(  
staff_id INT PRIMARY KEY,  
name VARCHAR(50)  
)
```

# CREATE TABLE

```
CREATE TABLE staff(  
staff_id SERIAL PRIMARY KEY,  
name VARCHAR(50)  
)
```

**SERIAL:** Creates an auto-increment sequence  
(typically used with primary key)

# CREATE TABLE

```
CREATE TABLE staff(  
staff_id SERIAL PRIMARY KEY,  
name VARCHAR(50) NOT NULL  
)
```

# CREATE TABLE

```
CREATE TABLE staff(  
staff_id SERIAL PRIMARY KEY,  
name VARCHAR(50) UNIQUE NOT NULL  
)
```

# CREATE TABLE

```
CREATE TABLE staff(  
staff_id SERIAL PRIMARY KEY,  
name VARCHAR(50) NOT NULL  
UNIQUE(name,staff_id)  
)
```

# CREATE TABLE

```
DROP TABLE <table_name>
```

# CREATE TABLE

```
DROP TABLE IF EXISTS <table_name>
```

# CREATE TABLE

```
DROP TABLE IF EXISTS directors
```

Data Output

Explain

Messages

Notifications

```
NOTICE: table "directors" does not exist, skipping  
DROP TABLE
```

# INSERT

```
INSERT INTO <table>
```

# INSERT

```
INSERT INTO <table>  
VALUES (value1,value2[,...])
```

transaction_id [PK] integer	customer_id integer	film_id integer	amount numeric (5,2)	promotion character varying (10)
--------------------------------	------------------------	--------------------	-------------------------	-------------------------------------

# INSERT

```
INSERT INTO online_sales  
VALUES (1, 269, 13, 10.99, 'BUNDLE2022')
```

transaction_id [PK] integer	customer_id integer	film_id integer	amount numeric (5,2)	promotion_code character varying (10)
1	1	269	13	10.99

# INSERT

```
INSERT INTO online_sales  
(customer_id, film_id, amount)  
VALUES (269, 13, 10.99)
```

transaction_id [PK] integer	customer_id integer	film_id integer	amount numeric (5,2)	promotion character varying (10)
1	1	269	13	10.99

SERIAL

DEFAULT

# INSERT

```
INSERT INTO online_sales  
(customer_id, amount)  
VALUES (269, 10.99)
```

transaction_id [PK] integer	customer_id integer	film_id integer	amount numeric (5,2)	promotion character varying (10)
1	1	269	[null]	10.99

SERIAL

DEFAULT

# INSERT

```
INSERT INTO online_sales  
(amount, customer_id)  
VALUES (10.99, 269)
```

transaction_id [PK] integer	customer_id integer	film_id integer	amount numeric (5,2)	promotion character varying (10)
1	1	269	[null]	10.99

SERIAL

DEFAULT

# INSERT

```
INSERT INTO online_sales  
(customer_id)  
VALUES (269)
```

Data Output Explain Messages Notifications

ERROR: null value in column "amount" of relation "online\_sales" violates not-null constraint  
DETAIL: Failing row contains (2, 269, null, null, None).  
SQL state: 23502

# INSERT

```
INSERT INTO online_sales  
(customer_id, film_id, amount)  
VALUES (269, 13, 10.99), (270, 12, 22.99)
```

	transaction_id [PK] integer	customer_id integer	film_id integer	amount numeric (5,2)	promotion character varying (10)
1	8	269	13	10.99	None
2	9	270	12	22.99	None

# INSERT

```
INSERT INTO online_sales  
(customer_id, film_id, amount)  
VALUES  
(269, 13, 10.99),  
(270, 12, 22.99)
```

	transaction_id [PK] integer	customer_id integer	film_id integer	amount numeric (5,2)	promotion character varying (10)
1	8	269	13	10.99	None
2	9	270	12	22.99	None

# ALTER TABLE

ADD, DELETE columns

ADD, DROP constraints

RENAME columns

ALTER data types

# ALTER TABLE

```
ALTER TABLE <table_name>  
ALTER_ACTION
```

# ALTER TABLE

```
ALTER TABLE <table_name>  
DROP COLUMN <column_name>
```

DROP

# ALTER TABLE

```
ALTER TABLE staff  
DROP COLUMN first_name
```

DROP

# ALTER TABLE

```
ALTER TABLE staff  
DROP COLUMN IF EXISTS first_name
```

DROP

# ALTER TABLE

```
ALTER TABLE <table_name>  
ADD COLUMN <column_name>
```

DROP

ADD

# ALTER TABLE

```
ALTER TABLE staff  
ADD COLUMN date_of_birth DATE
```

DROP

ADD

# ALTER TABLE

```
ALTER TABLE staff  
ADD COLUMN IF NOT EXISTS date_of_birth DATE
```

DROP

ADD

# ALTER TABLE

```
ALTER TABLE <table_name>  
ALTER COLUMN <column_name> TYPE NEW_TYPE
```

DROP

ADD

TYPE

# ALTER TABLE

```
ALTER TABLE staff  
ALTER COLUMN address_id TYPE SMALLINT
```

DROP

ADD

TYPE

# ALTER TABLE

```
ALTER TABLE <table_name>  
RENAME COLUMN <old_column_name> TO <new_column_name>
```

DROP

ADD

TYPE

RENAME

# ALTER TABLE

```
ALTER TABLE staff  
    RENAME COLUMN first_name TO name
```

DROP

ADD

TYPE

RENAME

# ALTER TABLE

```
ALTER TABLE <table_name>  
ALTER COLUMN <column_name> SET DEFAULT <value>
```

DROP

ADD

TYPE

RENAME

DEFAULT

# ALTER TABLE

```
ALTER TABLE <table_name>  
ALTER COLUMN <column_name> SET DEFAULT <value>
```

DROP

ADD

TYPE

RENAME

DEFAULT

# ALTER TABLE

```
ALTER TABLE staff  
ALTER COLUMN store_id SET DEFAULT 1
```

DROP

ADD

TYPE

RENAME

DEFAULT

# ALTER TABLE

```
ALTER TABLE <table_name>  
ALTER COLUMN <column_name> DROP DEFAULT
```

DROP

ADD

TYPE

RENAME

DEFAULT

# ALTER TABLE

```
ALTER TABLE <table_name>  
ALTER COLUMN <column_name> SET NOT NULL
```

DROP

ADD

TYPE

RENAME

DEFAULT

# ALTER TABLE

```
ALTER TABLE <table_name>  
ALTER COLUMN <column_name> DROP NOT NULL
```

DROP

ADD

TYPE

RENAME

DEFAULT

NOT NULL

# ALTER TABLE

```
ALTER TABLE <table_name>
ADD CONSTRAINT <constraint_name> UNIQUE(column1)
```

DROP

ADD

TYPE

RENAME

DEFAULT

NOT NULL

TABLE CONSTRAINT

# ALTER TABLE

```
ALTER TABLE <table_name>
ADD CONSTRAINT <constraint_name>
UNIQUE(column1, column2[, ...])
```

DROP

ADD

TYPE

RENAME

DEFAULT

NOT NULL

TABLE CONSTRAINT

# ALTER TABLE

```
ALTER TABLE <table_name>
ADD CONSTRAINT <constraint_name>,
ADD PRIMARY KEY(column1, column2[, ...])
```

DROP

ADD

TYPE

RENAME

DEFAULT

NOT NULL

TABLE CONSTRAINT

PRIMARY KEY

# ALTER TABLE

```
ALTER TABLE <table_name>
ADD CONSTRAINT <constraint_name>,
ADD PRIMARY KEY(column1, column2[, ...])
```

DROP

ADD

TYPE

RENAME

DEFAULT

NOT NULL

TABLE CONSTRAINT

PRIMARY KEY

# ALTER TABLE

```
ALTER TABLE director
ALTER COLUMN director_account_name SET DEFAULT 3,
ALTER COLUMN first_name TYPE TEXT,
ALTER COLUMN last_name TYPE TEXT,
ADD COLUMN middle_name TEXT,
ADD CONSTRAINT constraint_1 UNIQUE(account_name)
```

DROP

ADD

TYPE

RENAME

DEFAULT

NOT NULL

TABLE CONSTRAINT

PRIMARY KEY

# ALTER TABLE

```
ALTER TABLE director
ALTER COLUMN director_account_name SET DEFAULT 3,
ALTER COLUMN first_name TYPE TEXT,
ALTER COLUMN last_name TYPE TEXT,
ADD COLUMN middle_name TEXT,
ADD CONSTRAINT constraint_1 UNIQUE(account_name)
```

DROP

ADD

TYPE

RENAME

DEFAULT

NOT NULL

TABLE CONSTRAINT

PRIMARY KEY

# ALTER TABLE

```
ALTER TABLE old_table_name  
RENAME new_table_name
```

DROP

ADD

TYPE

RENAME

DEFAULT

NOT NULL

TABLE CONSTRAINT

PRIMARY KEY

# ALTER TABLE

```
ALTER TABLE director  
    RENAME director_table
```

DROP

ADD

TYPE

RENAME

DEFAULT

NOT NULL

TABLE CONSTRAINT

PRIMARY KEY

# CHECK

Limit the value range that can be placed in a column

# CHECK

```
CREATE TABLE <table_name> (  
    <column_name> TYPE CHECK(condition))
```

# CHECK

```
CREATE TABLE director (
    name TEXT CHECK (length(name)>1))
```

```
ERROR: new row for relation "director" violates check constraint "director_first_name_check"
DETAIL: Failing row contains (1, null, M, null, null, null).
SQL state: 23514
```

# CHECK

```
CREATE TABLE director(  
    name TEXT CONSTRAINT name_length CHECK (length(name)>1))
```

# CHECK

```
CREATE TABLE director (
    name TEXT CHECK (length(name)>1))
```

**Default name:** <table>\_<column>\_check

# CHECK

```
CREATE TABLE director (  
    name TEXT CHECK (length(name)>1))
```

**Default name:** <table>\_<column>\_check

**Default name:** director\_name\_check

# CHECK

```
CREATE TABLE director (
    name TEXT,
    date_of_birth DATE,
    start_date DATE,
    end_date DATE CHECK(start_date > '01-01-2000'))
```

**Default name:** director\_start\_date\_check

# CHECK

```
CREATE TABLE director (
    name TEXT,
    date_of_birth DATE,
    start_date DATE,
    end_date DATE CHECK(start_date > date_of_birth))
```

**Default name:** director\_check

# CHECK

```
CREATE TABLE director (
    name TEXT,
    date_of_birth DATE,
    start_date DATE,
    end_date DATE CHECK(start_date > date_of_birth))
```

```
INSERT INTO director
(date_of_birth,start_date)
VALUES ('01-01-1902','01-01-1900')
```

new row for relation "director" violates  
check constraint "director\_check"

# CHECK

```
ALTER TABLE director  
ADD CONSTRAINT date_check CHECK(start_date < end_date )
```

# CHECK

```
ALTER TABLE director  
DROP CONSTRAINT date_check
```

# CHECK

```
ALTER TABLE director  
RENAME CONSTRAINT date_check TO data_constraint
```

# CHECK

```
CREATE TABLE director (
    name TEXT,
    date_of_birth DATE CHECK(date_of_birth > '01-01-1900'))
```

# Challenge

Create a table called songs with the following columns:

<b>song_id</b> [PK] integer	<b>song_name</b> character varying (30)	<b>genre</b> character varying (30)	<b>price</b> numeric (4,2)	<b>release_date</b> date
--------------------------------	--	--	-------------------------------	-----------------------------

1. During creation add the DEFAULT 'Not defined' to the genre.
2. Add the not null constraint to the song\_name column
3. Add the constraint with default name to ensure the price is at least 1.99.
4. Add the constraint date\_check to ensure the release date is between today and 01-01-1950.
5. Try to insert a row like this:
6. Modify the constraint to be able to have 0.99 allowed as the lowest possible price.
7. Try again to insert the row.

<b>song_id</b> [PK] integer	<b>song_name</b> character varying (30)	<b>genre</b> character varying (30)	<b>price</b> numeric (4,2)	<b>release_date</b> date
1	4 SQL song	Not defined	0.99	2022-01-07

# Day 10



# UPDATE

```
UPDATE <table>  
SET <column>=value
```

# UPDATE

```
UPDATE <table>  
SET <column>=value
```

	song_id [PK] integer	song_name character varying (30)	genre character varying (30)	price numeric (4,2)	release_date date
1	2	SQL song	Not defined	0.99	2022-01-07
2	3	SQL song2	Not defined	0.99	2022-01-07
3	4	SQL song3	Not defined	0.99	2022-01-07

# UPDATE

```
UPDATE songs  
SET genre='Country music'
```

	song_id [PK] integer	song_name character varying (30)	genre character varying (30)	price numeric (4,2)	release_date date
1	2	SQL song	Not defined	0.99	2022-01-07
2	3	SQL song2	Not defined	0.99	2022-01-07
3	4	SQL song3	Not defined	0.99	2022-01-07

# UPDATE

```
UPDATE songs  
SET genre='Country music'
```

song_id [PK] integer	song_name character varying (30)	genre character varying (30)	price numeric (4,2)	release_date date
2	SQL song	Country music	0.99	2022-01-07
3	SQL song2	Country music	0.99	2022-01-07
4	SQL song3	Country music	0.99	2022-01-07

# UPDATE

```
UPDATE songs  
SET genre='Pop music'  
WHERE song_id=4
```

song_id [PK] integer	song_name character varying (30)	genre character varying (30)	price numeric (4,2)	release_date date
2	SQL song	Country music	0.99	2022-01-07
3	SQL song2	Country music	0.99	2022-01-07
4	SQL song3	Country music	0.99	2022-01-07

# UPDATE

```
UPDATE songs  
SET genre='Pop music'  
WHERE song_id=4
```

song_id [PK] integer	song_name character varying (30)	genre character varying (30)	price numeric (4,2)	release_date date
2	SQL song	Country music	0.99	2022-01-07
3	SQL song2	Country music	0.99	2022-01-07
4	SQL song3	Pop music	0.99	2022-01-07

# UPDATE

```
UPDATE songs  
SET price=song_id+0.99
```

song_id [PK] integer	song_name character varying (30)	genre character varying (30)	price numeric (4,2)	release_date date
2	SQL song	Country music	0.99	2022-01-07
3	SQL song2	Country music	0.99	2022-01-07
4	SQL song3	Pop music	0.99	2022-01-07

# UPDATE

```
UPDATE songs  
SET price=song_id+0.99
```

song_id [PK] integer	song_name character varying (30)	genre character varying (30)	price numeric (4,2)	release_date date
2	SQL song	Country music	2.99	2022-01-07
3	SQL song2	Country music	3.99	2022-01-07
4	SQL song3	Pop music	4.99	2022-01-07

# INSERT

```
INSERT INTO online_sales  
VALUES (1, 269, 13, 10.99, 'BUNDLE2022')
```

transaction_id [PK] integer	customer_id integer	film_id integer	amount numeric (5,2)	promotion_code character varying (10)
1	1	269	13	10.99

# INSERT

```
INSERT INTO online_sales  
(customer_id, film_id, amount)  
VALUES (269, 13, 10.99)
```

transaction_id [PK] integer	customer_id integer	film_id integer	amount numeric (5,2)	promotion character varying (10)
1	1	269	13	10.99

SERIAL

DEFAULT

# Challenge

Update all rental prices that are 0.99 to 1.99.

The *customer* table needs to be altered as well:

1. Add the column *initials* (data type varchar(10))
2. Update the values to the actual initials for example *Frank Smith* should be *F.S.*

# Challenge

Create a table called `users` with the following columns:

Data Output		Explain	Messages	Notifications
<code>user_id</code> [PK] integer	<code>first_name</code> text	<code>last_name</code> text	<code>user_name</code> text	<code>signup_date</code> date

1. During creation add the DEFAULT current\_date to the signup\_date.
2. Add the constraint `namelength` to ensure the user\_name has more than 2 characters.
3. Add the constraint with default name to ensure the birthdate is after 01-01-1900.
4. After the creation rename `namelength` to `name_length`.
5. Try to add Frank Smith with user name `franksmith1` and birthday `02-12-1905`.
6. Modify the constraint on the birthdate so that no dates after 01-01-1910 are allowed.
7. Try again to add Frank Smith with user name `franksmith1` and birthday `02-12-1905`.

# Challenge

During creation add the DEFAULT 'Not defined' to the genre.

2. Add the not null constraint to the song\_name column
3. Add the constraint with default name to ensure the price is at least 1.99.
4. Add the constraint *date\_check* to ensure the release date is between today and 01-01-1950.
5. Try to add Frank Smith with user name franksmith1 and birthday 02-12-1905.
6. Modify the constraint on the birthdate so that no dates after 01-01-1910 are allowed.
7. Try again to add Frank Smith with user name franksmith1 and birthday 02-12-1905.

# Constraints

## COLUMN CONSTRAINTS

What constraints do we have?

NOT NULL

Ensures that a column cannot have a NULL value

UNIQUE

Ensures that all values in a column are different

DEFAULT

Sets a default value for a column if no value is specified

```
ERROR: insert or update on table "director" violates foreign key constraint "director_address_id_fkey"
DETAIL: Key (address_id)=(0) is not present in table "address".
SQL state: 23503
```

REFERENCES

Ensures referential integrity (only values of another column can be used)

CHECK

Ensures that the values in a column satisfies a specific condition

# Constraints

TABLE  
CONSTRAINTS

What constraints do we have?

PRIMARY KEY ( column [ ... ] )

UNIQUE ( column [ ... ] )

CHECK ( search\_condition )

# ALTER TABLE

```
ALTER TABLE <table_name>  
ALTER COLUMN <column_name> SET DEFAULT <value>
```

DROP

ADD

TYPE

RENAME

DEFAULT

# ALTER TABLE

```
ALTER TABLE staff  
RENAME COLUMN first_name TO name,  
DROP COLUMN last_name
```

CONSTRAINT

# ALTER TABLE

```
ALTER TABLE <table_name>
ALTER COLUMN <column_name>
DROP DEFAULT <value>
```

DROP

ADD

ALTER TYPE

RENAME

DEFAULT

CONSTRAINT

# ALTER TABLE

```
ALTER TABLE staff  
ALTER COLUMN first_name TEXT
```

# DELETE

```
DELETE FROM <table>  
WHERE condition
```

# DELETE

```
DELETE FROM songs  
WHERE song_id=4
```

	song_id [PK] integer	song_name character varying (30)	genre character varying (30)	price numeric (4,2)	release_date date
1	2	SQL song	Not defined	0.99	2022-01-07
2	3	SQL song2	Not defined	0.99	2022-01-07
3	4	SQL song3	Not defined	0.99	2022-01-07

# DELETE

```
DELETE FROM songs  
WHERE song_id IN (3,4)
```

	song_id [PK] integer	song_name character varying (30)	genre character varying (30)	price numeric (4,2)	release_date date
1	2	SQL song	Not defined	0.99	2022-01-07
2	3	SQL song2	Not defined	0.99	2022-01-07
3	4	SQL song3	Not defined	0.99	2022-01-07

# DELETE

```
DELETE FROM songs
```

	song_id [PK] integer	song_name character varying (30)	genre character varying (30)	price numeric (4,2)	release_date date
1	2	SQL song	Not defined	0.99	2022-01-07
2	3	SQL song2	Not defined	0.99	2022-01-07
3	4	SQL song3	Not defined	0.99	2022-01-07

# DELETE

```
DELETE FROM songs  
WHERE song_id IN (3,4)  
RETURNING song_id
```

	song_id	[PK] integer
1		3
2		4

# DELETE

```
DELETE FROM songs  
WHERE song_id IN (3,4)  
RETURNING *
```

song_id [PK] integer	song_name character varying (30)	genre character varying (30)	price numeric (4,2)	release_date date
3	SQL song2	Country music	3.99	2022-01-07
4	SQL song3	Pop music	4.99	2022-01-07

# CREATE TABLE ... AS

```
CREATE TABLE <table_name>  
AS query
```

# CREATE TABLE ... AS

```
CREATE TABLE customer_test  
AS  
SELECT * FROM customer
```

# CREATE TABLE ... AS

```
CREATE TABLE customer_anonymous  
AS  
SELECT customer_id, initials  
FROM customer  
WHERE first_name LIKE 'C%'
```

# CREATE TABLE ... AS

```
CREATE TABLE customer_anonymous  
AS  
SELECT customer_id, initials  
FROM customer  
WHERE first_name LIKE 'C%'
```

```
SELECT * FROM customer_anonymous
```

Physical storage needed!

Data can change!

Alternative: Create a view and just store the statement!

# CREATE VIEW ... AS

```
CREATE VIEW <view_name>  
AS query
```

# CREATE VIEW ... AS

```
CREATE VIEW customer_anonymous  
AS  
SELECT customer_id, initials  
FROM customer  
WHERE first_name LIKE 'C%'
```

# CREATE VIEW ... AS

```
CREATE VIEW customer_anonymous  
AS  
SELECT customer_id, initials  
FROM customer  
WHERE first_name LIKE 'C%'
```

```
SELECT * FROM customer_anonymous
```

# CREATE VIEW ... AS

If the query is slow the view will be slow!

```
CREATE TABLE customer_an_table  
AS  
SELECT * FROM customer_anonymous
```

Problem: That table will not be updated if data in the underlying tables change!

# CREATE VIEW ... AS

If the query is slow the view will be slow.

Data Output Explain Messages Notifications

	customer_id integer	name text	address text	postal_code text	phone text	city text	country text
1	1	MARY BROWN	1913 Hanoi Way	35200	28303384290	Sasebo	Japan
2	2	PATRICIA JOHNSON	1121 Loja Avenue	17886	838635286649	San Bernardino	United States
3	3	LINDA WILLIAMS	692 Joliet Street	83579	448477190408	Athenai	Greece

Problem: That table will not be updated if data in the underlying tables change!

# Managing views

ALTER VIEW

ALTER MATERIALIZED VIEW

DROP VIEW

DROP MATERIALIZED VIEW

CREATE OR REPLACE VIEW

X

# DROP VIEW

```
DROP VIEW customer_anonymous
```

```
DROP MATERIALIZED VIEW customer_anonymous
```

# ALTER VIEW

```
ALTER VIEW customer_anonymous  
RENAME TO v_customer_info
```

```
ALTER VIEW v_customer_info  
RENAME COLUMN name TO customer_name
```

# REPLACE VIEW

```
CREATE OR REPLACE VIEW v_customer_info  
AS new_query
```

Not possible with  
MATERIALIZED VIEW!

# CREATE VIEW ... AS

If the query is slow the view will be slow!

```
CREATE TABLE customer_an_table  
AS  
SELECT * FROM customer_anonymous
```

Problem: That table will not be updated if data in the underlying tables change!

# VIEW

```
UPDATE songs  
SET genre='Country music'
```

	song_id [PK] integer	song_name character varying (30)	genre character varying (30)	price numeric (4,2)	release_date date
1	2	SQL song	Not defined	0.99	2022-01-07
2	3	SQL song2	Not defined	0.99	2022-01-07
3	4	SQL song3	Not defined	0.99	2022-01-07

# Import & Export

Import external data into an existing table

Export data from a table into a csv file

Table needs to be created first!

Data needs to be in correct format!

# Day 11



# Window functions

```
SELECT  
transaction_id,  
payment_type,  
customer_id,  
price_in_transaction,  
(SELECT SUM(price_in_transaction)  
FROM sales s2  
WHERE s2.customer_id=s1.customer_id)  
FROM sales s1
```

price_in_transaction
18.29
1.49
5.89
11.59

No. of rows not affected

Aggregated by

transaction_id	payment_type	customer_id	price_in_transaction	total_spent_by_customer
1	visa	4	18.29	6182.79
2	visa	5	1.49	8762.30
3	visa	5	5.89	8762.30
4	mastercard	8	11.59	5779.96

# Window functions

```
SELECT  
transaction_id,  
payment_type,  
customer_id,  
price_in_transaction,  
SUM(price_in_transaction) OVER(PARTITION BY customer_id)  
FROM sales s
```

No. of rows not affected

Aggregated by

transaction_id	payment_type	customer_id	price_in_transaction	total_spent_by_customer
1	visa	4	18.29	6182.79
2	visa	5	1.49	8762.30
3	visa	5	5.89	8762.30
4	mastercard	8	11.59	5779.96

# Window functions

```
SELECT  
transaction_id,  
payment_type,  
customer_id,  
price_in_transaction,  
COUNT(*) OVER(PARTITION BY customer_id)  
FROM sales s
```

No. of rows not affected

Aggregated by

transaction_id [PK] integer	payment_type character varying (20)	customer_id integer	price_in_transaction numeric (5,2)	no_of_transactions_by_customer bigint
1	visa	4	18.29	541
2	visa	5	1.49	770
3	visa	5	5.89	770
4	mastercard	8	11.59	514

# Window functions

```
SELECT  
transaction_id,  
payment_type,  
customer_id,  
price_in_transaction,  
COUNT(*) OVER(PARTITION BY payment_type)  
FROM sales s
```

No. of rows not affected

Aggregated by

transaction_id [PK] integer	payment_type character varying (20)	customer_id integer	price_in_transaction numeric (5,2)	no_of_transactions_by_type bigint
1	visa	4	18.29	1398
2	visa	5	1.49	1398
3	visa	5	5.89	1398
4	mastercard	8	11.59	1420
5	mastercard	5	12.39	1420

# Window functions

No. of rows not affected

Aggregated by

transaction_id [PK] integer	payment_type character varying (20)	customer_id integer	price_in_transaction numeric (5,2)	no_of_transactions_by_type bigint
1	visa	4	18.29	1398
2	visa	5	1.49	1398
3	visa	5	5.89	1398
4	mastercard	8	11.59	1420
5	mastercard	5	12.39	1420

# OVER()

No. of rows not affected

Aggregated by

transaction_id [PK] integer	payment_type character varying (20)	customer_id integer	price_in_transaction numeric (5,2)	no_of_transactions_by_type bigint	
1	visa	4	18.29	1398	
2	visa	5	1.49	1398	
3	visa	5	5.89	1398	
4	mastercard	8	11.59	1420	
5	mastercard	5	12.39	1420	

SUM()

COUNT()

RANK()

FIRST\_VALUE()

LEAD()

LAG()

# Window functions

```
AGG(agg_column) OVER(PARTITION BY partition_column)
```

Window

# Challenge

Write a query that returns the list of movies including

- film\_id,
- title,
- length,
- category,
- average length of movies in that category.

Order the results by film\_id.

Result

film_id	title	category	length_of_movie	avg_length_in_category
1	ACADEMY DINOSAUR	Documentary	86	108.75
2	ACE GOLDFINGER	Horror	48	112.48
3	ADAPTATION HOLES	Documentary	50	108.75
4	AFFAIR PREJUDICE	Horror	117	112.48
5	AFRICAN EGG	Family	130	114.78

# Challenge

- Write a query that returns all payment details including
    - the number of payments that were made by this customer and that amount
- Order the results by payment\_id.

Result

payment_id	customer_id	staff_id	rental_id	amount	payment_date	no_payments_with_that_amount
integer	smallint	smallint	integer	numeric (5,2)	timestamp with time	bigint
16050	269	2	7	1.99	2020-01-24 22:40:...	1
16051	269	1	98	0.99	2020-01-25 16:16:...	3
16052	269	2	678	6.99	2020-01-28 22:44:...	5

# Challenge

Write a query that returns the running total of how late the flights are (difference between actual\_arrival and scheduled arrival) ordered by flight\_id including the departure airport.

As a second query, calculate the same running total but partition also by the departure airport.

Result

flight_id [PK] integer	departure_airport character (3)	sum interval
1	DME	00:09:00
2	DME	00:10:00
3	DME	00:14:00
4	DME	00:14:00

flight_id [PK] integer	departure_airport character (3)	sum interval
20981	AAQ	00:02:00
20982	AAQ	00:04:00
20983	AAQ	00:04:00

# Challenge

Write a query that returns the customers' name, the country and how many payments they have. For that use the existing view *customer\_list*.

name text	country text	count bigint
RICARDO MEADOR	Japan	21
NANCY THOMAS	India	28
THELMA MURRAY	Peru	32

Afterwards create a ranking of the top customers with most sales for each country. Filter the results to only the top 3 customers per country.

## Result

name text	country text	count bigint	rank bigint
VERA MCCOY	Afghanistan	18	1
JUNE CARROLL	Algeria	37	1
MARIO CHEATHAM	Algeria	28	2
JUDY GRAY	Algeria	25	3

# Challenge

Write a query that returns the revenue of the day and the revenue of the previous day.

sum numeric	day date	previous_day numeric	difference numeric
62.86	2020-01-24	[null]	[null]
563.64	2020-01-25	62.86	500.78
736.30	2020-01-26	563.64	172.66

Afterwards calculate also the percentage growth compared to the previous day.

## Result

sum numeric	day date	previous_day numeric	difference numeric	percentage_growth numeric
62.86	2020-01-24	[null]	[null]	[null]
563.64	2020-01-25	62.86	500.78	796.66
736.30	2020-01-26	563.64	172.66	30.63
707.29	2020-01-27	736.30	-29.01	-3.94

# Challenge

Write a query that calculates now the share of revenue each staff\_id makes per customer. The result should look like this:

first_name text	last_name text	staff_id smallint	total numeric	percentage numeric
AARON	SELBY	[null]	110.76	100.00
AARON	SELBY	1	63.86	57.66
AARON	SELBY	2	46.90	42.34
ADAM	GOOCH	[null]	101.78	100.00
ADAM	GOOCH	1	51.89	50.98
ADAM	GOOCH	2	49.89	49.02

Hint

You need to use a subquery.

# Day 12



# CUBE & ROLLUP

GROUP BY

CUBE (column1, column2, column3)

GROUP BY

GROUPING SETS (

(column1, column2, column3),  
(column1, column2),  
(column1, column3),  
(column2, column3),  
(column1),  
(column2),  
(column3),  
(  
))

# CUBE & ROLLUP

GROUP BY

ROLLUP (column1, column2, column3)

GROUP BY

GROUPING SETS (

(column1, column2, column3),

(column1, column2),

(column1),

()

)

Hierarchy

# Challenge

Write a query that calculates a booking amount rollup for the hierarchy of quarter, month, week in month and day.

quarter  numeric	month  numeric	week_in_month  text	day  date	booking_amount  numeric
2	6	3	2017-06-21	441900.00
2	6	3	[null]	441900.00
2	6	4	2017-06-22	775300.00
2	6	4	2017-06-23	1822000.00

Hint

Pattern for week in month is 'w'.

# Self-join

Referencing employee\_id



employee_id	name	manager_id
1	Liam Smith	[null]
2	Oliver Brown	1
3	Elijah Jones	1
4	William Miller	1
5	James Davis	2
6	Olivia Hernandez	2

# Self-join

Referencing employee\_id



employee_id	name	manager_id	employee_id
1	Liam Smith	[null]	[null]
2	Oliver Brown	1	1
3	Elijah Jones	1	1
4	William Miller	1	1
5	James Davis	2	2
6	Olivia Hernandez	2	2

# Self-join

Referencing employee\_id



employee_id	name	manager_id	employee_id	manager
integer	character varying (50)	integer	integer	character varying (50)
1	Liam Smith	[null]	[null]	[null]
2	Oliver Brown	1	1	Liam Smith
3	Elijah Jones	1	1	Liam Smith
4	William Miller	1	1	Liam Smith
5	James Davis	2	2	Oliver Brown
6	Olivia Hernandez	2	2	Oliver Brown

# Self-join

Standard join with itself

```
SELECT  
t1.column1,  
t2.column1  
[,...]  
FROM table1 t1  
LEFT JOIN table1 t2  
ON t1.column1=t2.column1
```

# Self-join

employee_id	employee	manager	manager_id
integer	character varying (50)	character varying (50)	integer
1	Liam Smith	[null]	[null]
2	Oliver Brown	Liam Smith	1

```
SELECT
t1.column1,
t2.column1
[,...]
FROM employee t1
LEFT JOIN employee t2
ON t1.column1=t2.column1
```

# Self-join

employee_id	employee	manager	manager_id
integer	character varying (50)	character varying (50)	integer
1	Liam Smith	[null]	[null]
2	Oliver Brown	Liam Smith	1

```
SELECT
t1.column1,
t2.column1
[,...]
FROM employee emp
LEFT JOIN employee mng
ON t1.column1=t2.column1
```

# Self-join

employee_id	employee	manager	manager_id
1	Liam Smith	[null]	[null]
2	Oliver Brown	Liam Smith	1

```
SELECT
t1.column1,
t2.column1
[,...]
FROM employee emp
LEFT JOIN employee mng
ON emp.manager_id=t2.column1
```

# Self-join

employee_id	employee	manager	manager_id
integer	character varying (50)	character varying (50)	integer
1	Liam Smith	[null]	[null]
2	Oliver Brown	Liam Smith	1

```
SELECT
t1.column1,
t2.column1
[,...]
FROM employee emp
LEFT JOIN employee mng
ON emp.manager_id=mng.employee_id
```

# Self-join

employee_id	employee	manager	manager_id
1	Liam Smith	[null]	[null]
2	Oliver Brown	Liam Smith	1

```
SELECT
emp.employee_id,
t2.column1
[,...]
FROM employee emp
LEFT JOIN employee mng
ON emp.manager_id=mng.employee_id
```

# Self-join

employee_id	employee	manager	manager_id
1	Liam Smith	[null]	[null]
2	Oliver Brown	Liam Smith	1

```
SELECT  
emp.employee_id,  
emp.name AS employee  
[,...]  
FROM employee emp  
LEFT JOIN employee mng  
ON emp.manager_id=mng.employee_id
```

# Self-join

employee_id	employee	manager	manager_id
1	Liam Smith	[null]	[null]
2	Oliver Brown	Liam Smith	1

```
SELECT  
    emp.employee_id,  
    emp.name AS employee,  
    mng.name AS manager  
FROM employee emp  
LEFT JOIN employee mng  
ON emp.manager_id=mng.employee_id
```

# Challenge

Find all the pairs of films with the same length!

title text	title text	length smallint
MUSCLE BRIGHT	SOLDIERS EVOLUTION	185
MUSCLE BRIGHT	HOME PITY	185
MUSCLE BRIGHT	DARN FORRESTER	185
MUSCLE BRIGHT	GANGS PRIDE	185
MUSCLE BRIGHT	POUND SEATTLE	185

title text	title text	length smallint
SATURN NAME	SATURN NAME	192



# CROSS JOIN

Cartesian product

All combinations of rows

CROSS JOIN

letter
A
B

number
1
2
3



number	letter
1	A
2	A
3	A
1	B
2	B
3	B

# CROSS JOIN

Cartesian product

All combinations of rows

CROSS JOIN

letter
A
B

number
1
1
3



number	letter
1	A
1	A
3	A
1	B
1	B
3	B

# CROSS JOIN

```
SELECT  
    t1.column1,  
    t2.column1  
FROM table1 t1  
CROSS JOIN table2 t2
```

# NATURAL JOIN

Just like a normal JOIN

Automatically joins using columns  
with the same column name

```
SELECT
*
FROM payment
NATURAL LEFT JOIN customer
```

# NATURAL JOIN

Just like a normal JOIN

Automatically joins using columns  
with the same column name

```
SELECT
*
FROM payment
NATURAL INNER JOIN customer
```

# Self-join

employee_id	employee	manager	manager_id
1	Liam Smith	[null]	[null]
2	Oliver Brown	Liam Smith	1

```
SELECT
    emp.employee_id,
    emp.name AS employee
    [,...]
FROM employee emp
LEFT JOIN employee mng
ON emp.manager_id=mng.employee_id
```

# CROSS JOIN

Cartesian product

All combinations of rows

CROSS JOIN

letter
A
B

number
1
1
3



number	letter
1	A
1	A
3	A
1	B
1	B
3	B

# Day 14



# USER-DEFINED FUNCTIONS

Extend functionality

Complex custom calculations

Safe languages

Unsafe languages

SQL

Python

PL/pgSQL

C

Extended version of SQL

# USER-DEFINED FUNCTIONS

```
CREATE FUNCTION <function_name> (param1, param2,...)
```

# USER-DEFINED FUNCTIONS

```
CREATE OR REPLACE FUNCTION <function_name> (param1,  
param2,...)
```

# USER-DEFINED FUNCTIONS

```
CREATE FUNCTION <function_name> (param1, param2,...)
```

# USER-DEFINED FUNCTIONS

```
CREATE FUNCTION <function_name> (param1, param2,...)  
RETURNS return_datatype
```

# USER-DEFINED FUNCTIONS

```
CREATE FUNCTION <function_name> (param1, param2,...)
RETURNS return_datatype
LANGUAGE plpgsql [sql|c|...]
```

# USER-DEFINED FUNCTIONS

```
CREATE FUNCTION <function_name> (param1, param2,...)
RETURNS return_datatype
LANGUAGE plpgsql [sql|c|...]
```

AS

\$\$

\$\$

# USER-DEFINED FUNCTIONS

```
CREATE FUNCTION <function_name> (param1, param2,...)
    RETURNS return_datatype
    LANGUAGE plpgsql [sql|c|...]
AS
$$
DECLARE
<variable declaration>;
$$
```

# USER-DEFINED FUNCTIONS

```
SELECT first_func(3,4)
```

$$3+4 + 3$$

first_func	integer
1	10

```
CREATE FUNCTION <function_name> (param1, param2,...)
    RETURNS return_datatype
    LANGUAGE plpgsql [sql|c|...]
AS
$$
DECLARE
<variable declaration>;
BEGIN
<function_definition>;
END;
$$
```

# USER-DEFINED FUNCTIONS

```
CREATE FUNCTION first_funct(param1, param2,...)
RETURNS return_datatype
LANGUAGE plpgsql [sql|c|...]
AS
$$
DECLARE
<variable declaration>;
BEGIN
<function_definition>;
END;
$$
```

# USER-DEFINED FUNCTIONS

```
CREATE FUNCTION first_funct(c1 INT, c2 INT)
    RETURNS return_datatype
    LANGUAGE plpgsql [sql|c|...]
AS
$$
DECLARE
<variable declaration>;
BEGIN
<function_definition>;
END;
$$
```

# USER-DEFINED FUNCTIONS

```
CREATE FUNCTION first_funct(c1 INT, c2 INT)
    RETURNS INT
    LANGUAGE plpgsql [sql|c|...]
AS
$$
DECLARE
<variable declaration>;
BEGIN
<function_definition>;
END;
$$
```

# USER-DEFINED FUNCTIONS

```
CREATE FUNCTION first_funct(c1 INT, c2 INT)
    RETURNS INT
    LANGUAGE plpgsql
AS
$$
DECLARE
<variable declaration>;
BEGIN
<function_definition>;
END;
$$
```

# USER-DEFINED FUNCTIONS

```
CREATE FUNCTION first_funct(c1 INT, c2 INT)
    RETURNS INT
    LANGUAGE plpgsql
AS
$$
DECLARE
c3 INT;
BEGIN
<function_definition>;
END;
$$
```

# USER-DEFINED FUNCTIONS

```
CREATE FUNCTION first_funct(c1 INT, c2 INT)
    RETURNS INT
    LANGUAGE plpgsql
AS
$$
DECLARE
c3 INT;
BEGIN
SELECT c1+c2+3;
END;
$$
```

# USER-DEFINED FUNCTIONS

```
CREATE FUNCTION first_funct(c1 INT, c2 INT)
    RETURNS INT
    LANGUAGE plpgsql [sql|c|...]
AS
$$
DECLARE
c3 INT;
BEGIN
SELECT c1+c2+3
INTO c3;
RETURN c3;
END;
$$
```

# USER-DEFINED FUNCTIONS

```
CREATE FUNCTION first_funct(c1 INT, c2 INT)
RETURNS INT
LANGUAGE plpgsql
AS
$$
DECLARE
BEGIN
RETURN SELECT c1+c2+3;
END ;
$$
```

# USER-DEFINED FUNCTIONS

```
CREATE FUNCTION first_funct(c1 INT, c2 INT)
RETURNS INT
LANGUAGE plpgsql
AS
$$
DECLARE
c3 INT;
BEGIN
SELECT c1+c2+3
INTO c3
FROM table;
RETURN c3;
END;
```

# Challenge

Create a function that expects the customer's first and last name and returns the total amount of payments this customer has made.

```
20  SELECT name_search('AMY', 'LOPEZ')
```

```
21 |
```

```
22 |
```

Data Output   Explain   Messages   Notifications

	name_search
◀	numeric
1	127.71

# TRANSACTIONS

Unit of work

One or multiple operations

# TRANSACTIONS

Bank transfer

<b>id</b> [PK] integer	<b>first_name</b> text	<b>last_name</b> text	<b>amount</b> numeric (9,2)
1	Tim	Brown	2500.00
2	Sandra	Miller	1600.00

100

One unit of work

# TRANSACTIONS

```
BEGIN TRANSACTION;
```

# TRANSACTIONS

```
BEGIN WORK;
```

# TRANSACTIONS

```
BEGIN;
```

# TRANSACTIONS

```
BEGIN;
```

```
OPERATION1;  
OPERATION2;
```

Not visible in other sessions  
(e.g. other users)

# TRANSACTIONS

```
BEGIN;
```

```
OPERATION1;  
OPERATION2;
```

```
COMMIT;
```

Not visible in other sessions  
(e.g. other users)

# TRANSACTIONS

```
BEGIN;
```

```
OPERATION1;  
OPERATION2;
```

```
COMMIT;
```

# Challenge

The two employees Miller McQuarter and Christalle McKenny have agreed to swap their positions incl. their salary.

emp_id [PK] integer	first_name	last_name	position_title	salary numeric (8,2)
1	Morrie	Conaboy	CTO	21268.94
2	Miller	McQuarter	Head of BI	14614.00
3	Christalle	McKenny	Head of Sales	12587.00

# ROLLBACK

```
BEGIN;
```

```
OPERATION1;  
OPERATION2;
```

```
COMMIT;
```

# ROLLBACK

```
BEGIN;  
  
OPERATION1;  
OPERATION2;  
ROLLBACK;  
COMMIT;
```

Undo everything in the current transaction that has not been committed yet!

# ROLLBACK

```
BEGIN;  
  
OPERATION1;  
OPERATION2;  
OPERATION3;  
OPERATION4;  
  
ROLLBACK;  
COMMIT;
```

# ROLLBACK

```
BEGIN;  
  
OPERATION1;  
OPERATION2;  
SAVEPOINT op2;  
OPERATION3;  
OPERATION4;  
  
ROLLBACK TO SAVEPOINT op2;  
COMMIT;
```

# ROLLBACK

```
BEGIN;  
  
OPERATION1;  
OPERATION2;  
SAVEPOINT op2;  
OPERATION3;  
SAVEPOINT op3;  
OPERATION4;  
  
ROLLBACK TO SAVEPOINT op3;  
COMMIT;
```

**Savepoints work only within a current transaction**

# ROLLBACK

```
BEGIN;  
  
OPERATION1;  
OPERATION2;  
SAVEPOINT op2;  
OPERATION3;  
SAVEPOINT op3;  
OPERATION4;  
  
RELEASE SAVEPOINT op3;  
COMMIT;
```

ROLLBACK;  
ends transaction

ROLLBACK TO SAVEPOINT;  
does not end transaction

Deleting a savepoint

# STORED PROCEDURES

User-defined function:  
CREATE FUNCTION

⚡ Downside:  
They cannot execute transactions

BEGIN;

COMMIT;

ROLLBACK;

✓ Stored procedures:  
They support transactions

# STORED PROCEDURES

```
CREATE PROCEDURE <procedure_name> (param1, param2,...)
```

# STORED PROCEDURES

```
CREATE OR REPLACE PROCEDURE <procedure_name> (param1,  
param2,...)
```

# STORED PROCEDURES

```
CREATE PROCEDURE <procedure_name> (param1, param2,...)
```

# STORED PROCEDURES

```
CREATE PROCEDURE <procedure_name> (param1, param2,...)
    LANGUAGE plpgsql [sql|c|...]
AS
$$
DECLARE
<variable declaration>
BEGIN
<procedure_definition>
END;
$$
```

# STORED PROCEDURES

```
CREATE FUNCTION <function_name> (param1, param2,...)
    RETURNS INT
    LANGUAGE plpgsql [sql|c|...]
AS
$$
DECLARE
<variable declaration>
BEGIN
<function_definition>
RETURN expression;
END;
$$
```

**Not possible in a  
stored procedure!**

# STORED PROCEDURES

```
CREATE PROCEDURE <procedure_name> (param1, param2,...)
    LANGUAGE plpgsql [sql|c|...]
AS
$$
DECLARE
<variable declaration>
BEGIN
<procedure_definition>
END;
$$
```

**Not possible in a  
stored procedure!**

# STORED PROCEDURES

```
CREATE PROCEDURE <procedure_name> (param1, param2,...)
    LANGUAGE plpgsql [sql|c|...]
AS
$$
DECLARE
<variable declaration>
BEGIN
<procedure_definition>
RETURN;
END;
$$
```

**Not possible in a  
stored procedure!**

# TRANSACTIONS

Bank transfer

<b>id</b> [PK] integer	<b>first_name</b> text	<b>last_name</b> text	<b>amount</b> numeric (9,2)
1	Tim	Brown	2500.00
2	Sandra	Miller	1600.00

100

One unit of work

# STORED PROCEDURES

```
CREATE PROCEDURE sp_transfer (tr_amount INT, sender INT, recipient INT)
    LANGUAGE plpgsql
AS
$$
```

# STORED PROCEDURES

```
CREATE PROCEDURE sp_transfer (tr_amount INT, sender INT, recipient INT)
    LANGUAGE plpgsql
AS
$$
BEGIN
    -- subtract from sender's balance
    UPDATE acc_balance
    SET amount = amount - tr_amount
    WHERE id = sender;
```

# STORED PROCEDURES

```
CREATE PROCEDURE sp_transfer (tr_amount INT, sender INT, recipient INT)
    LANGUAGE plpgsql
AS
$$
BEGIN
    -- subtract from sender's balance
    UPDATE acc_balance
    SET amount = amount - tr_amount
    WHERE id = sender;

    -- add to recipient's balance
    UPDATE acc_balance
    SET amount = amount + tr_amount
    WHERE id = recipient;
```

# STORED PROCEDURES

```
CREATE PROCEDURE sp_transfer (tr_amount INT, sender INT, recipient INT)
    LANGUAGE plpgsql
AS
$$
BEGIN
    -- subtract from sender's balance
    UPDATE acc_balance
    SET amount = amount - tr_amount
    WHERE id = sender;

    -- add to recipient's balance
    UPDATE acc_balance
    SET amount = amount + tr_amount
    WHERE id = recipient;

    COMMIT;
END;
$$
```

# STORED PROCEDURES

```
CALL <store_procedure_name> (param1, param2,...);
```

# STORED PROCEDURES

```
CALL <store_procedure_name> (param1, param2,...);
```

```
sp_transfer (tr_amount, sender,recipient)
```

```
CALL sp_transfer (100, 1,2);
```

# Challenge

Create a stored procedure called emp\_swap that accepts two parameters emp1 and emp2 as input and swaps the two employees' position and salary.

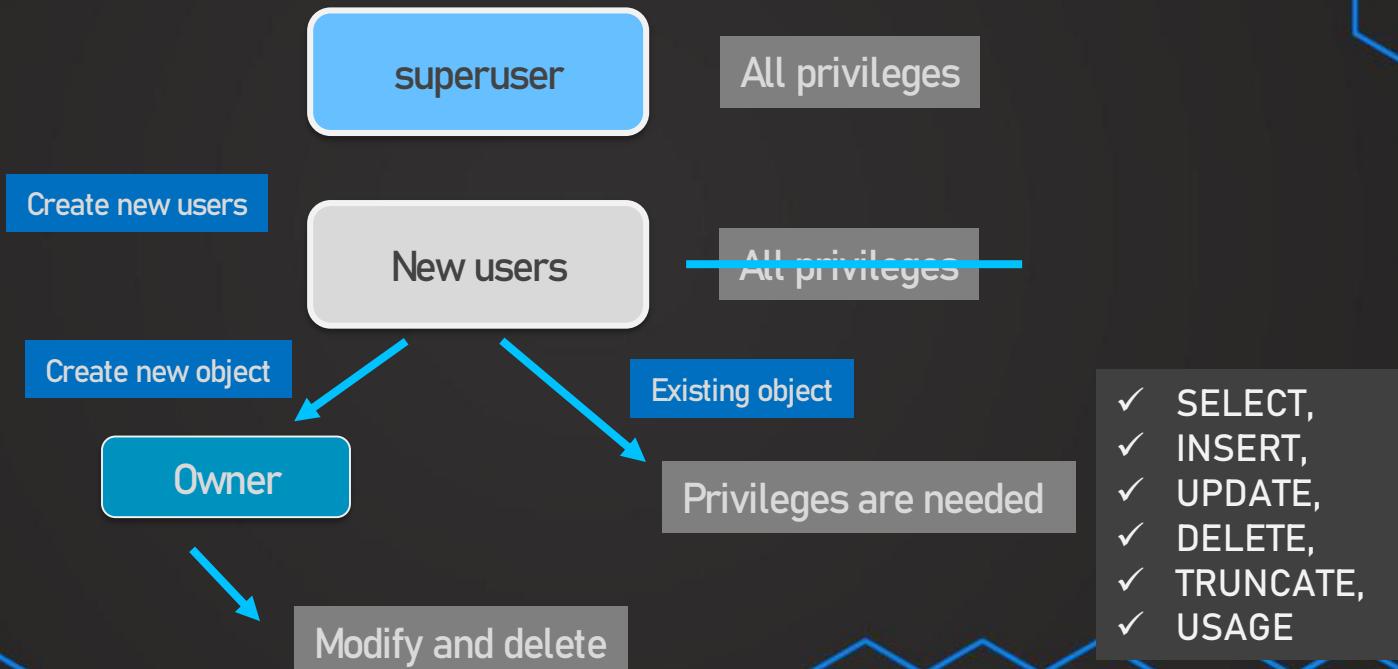
Test the stored procedure with emp\_id 2 and 3.

emp_id [PK] integer	first_name	last_name	position_title	salary numeric (8,2)
1	Morrie	Conaboy	CTO	21268.94
2	Miller	McQuarter	Head of BI	14614.00
3	Christalle	McKenny	Head of Sales	12587.00

# Day 15



# User management



# User management

Create new users

New users

Assign

Create new roles

role

Assign

Assign

- ✓ SELECT,
- ✓ INSERT,
- ✓ UPDATE,
- ✓ DELETE,
- ✓ TRUNCATE,
- ✓ USAGE

- ✓ SELECT,
- ✓ INSERT,
- ✓ UPDATE,
- ✓ DELETE,
- ✓ TRUNCATE,
- ✓ USAGE

# User management

Create new users

Sarah

Frank

Assign

Create new roles

Analyst

Assign

✓ SELECT,  
✓ USAGE

# CREATE USER

```
CREATE USER <user_name>  
WITH PASSWORD 'pwd123'
```



# CREATE USER

```
CREATE USER <user_name>  
WITH PASSWORD 'pwd123'
```

Data Output Explain **Messages** Notifications

CREATE ROLE

Query returned successfully in 67 msec.

Role

=

User + Login

# CREATE USER

```
CREATE USER <user_name>  
WITH PASSWORD 'pwd123'
```

```
CREATE ROLE <role_name>  
WITH LOGIN PASSWORD 'pwd123'
```

Data Output Explain **Messages** Notifications

CREATE ROLE

Query returned successfully in 67 msec.

Role

=

User + Login

# CREATE USER

```
DROP USER <user_name>
```

```
DROP ROLE <role_name>
```

# Privileges

- ✓ SELECT,
- ✓ INSERT,
- ✓ UPDATE,
- ✓ DELETE,
- ✓ TRUNCATE,
- ✓ USAGE,
- ✓ ALL

Privileges

```
GRANT privilege  
ON database_object  
TO USER | ROLE | PUBLIC
```

# Privileges

- ✓ SELECT,
- ✓ INSERT,
- ✓ UPDATE,
- ✓ DELETE,
- ✓ TRUNCATE,
- ✓ USAGE,
- ✓ ALL

Privileges

```
GRANT SELECT  
ON customer  
TO nikolai
```

# Privileges

- ✓ SELECT,
- ✓ INSERT,
- ✓ UPDATE,
- ✓ DELETE,
- ✓ TRUNCATE,
- ✓ USAGE,
- ✓ ALL

TABLES

```
GRANT SELECT  
ON customer  
TO nikolai
```

# Privileges

- ✓ SELECT,
- ✓ INSERT,
- ✓ UPDATE,
- ✓ DELETE,
- ✓ TRUNCATE,
- ✓ USAGE,
- ✓ ALL

TABLES

```
GRANT SELECT  
ON ALL TABLES IN SCHEMA schema_name  
TO nikolai
```

# Privileges

- ✓ SELECT,
- ✓ INSERT,
- ✓ UPDATE,
- ✓ DELETE,
- ✓ TRUNCATE,
- ✓ USAGE,
- ✓ ALL

TABLES

```
GRANT ALL  
ON ALL TABLES IN SCHEMA schema_name  
TO nikolai
```

# Privileges

- ✓ SELECT,
- ✓ INSERT,
- ✓ UPDATE,
- ✓ DELETE,
- ✓ TRUNCATE,
- ✓ USAGE,
- ✓ ALL

TABLES

superuser

owner

```
GRANT ALL  
ON ALL TABLES IN SCHEMA schema_name  
TO nikolai
```

# Privileges

- ✓ SELECT,
- ✓ INSERT,
- ✓ UPDATE,
- ✓ DELETE,
- ✓ TRUNCATE,
- ✓ USAGE,
- ✓ ALL

TABLES

superuser

owner

```
GRANT SELECT  
ON ALL TABLES IN SCHEMA schema_name  
TO nikolai WITH GRANT OPTION
```

# Privileges

- ✓ SELECT,
- ✓ INSERT,
- ✓ UPDATE,
- ✓ DELETE,
- ✓ TRUNCATE,
- ✓ USAGE,
- ✓ ALL

Privileges

```
REVOKE privilege  
ON database_object  
FROM USER | ROLE | PUBLIC
```

# Privileges

- ✓ SELECT,
- ✓ INSERT,
- ✓ UPDATE,
- ✓ DELETE,
- ✓ TRUNCATE,
- ✓ USAGE,
- ✓ ALL

Privileges

```
REVOKE privilege  
ON database_object  
FROM USER | ROLE | PUBLIC  
GRANTED BY USER | ROLE
```

# Privileges

- ✓ SELECT,
- ✓ INSERT,
- ✓ UPDATE,
- ✓ DELETE,
- ✓ TRUNCATE,
- ✓ USAGE,
- ✓ ALL

Privileges

```
REVOKE GRANT OPTION FOR privilege  
ON database_object  
FROM USER | ROLE | PUBLIC  
GRANTED BY USER | ROLE
```

# Privileges

Privilege	Applicable Object Types
SELECT	TABLE (and table-like objects), table column
INSERT	TABLE, table column
UPDATE	TABLE, table column
DELETE	TABLE
TRUNCATE	TABLE
CREATE	DATABASE, SCHEMA
CONNECT	DATABASE
EXECUTE	FUNCTION, PROCEDURE
USAGE	SCHEMA

# Privileges

How to grant acces?  
Typical statements

CREATE USER

```
CREATE USER amar  
WITH PASSWORD 'amar1234';
```

GRANT USAGE on schema

```
GRANT USAGE  
ON SCHEMA name  
TO amar;
```

GRANT SELECT & UPDATE

```
GRANT SELECT, UPDATE  
ON customer  
TO amar;
```

# Privileges

How to grant acces?  
Typical statements

GRANT all privileges on schema

```
GRANT ALL  
ON ALL TABLES IN SCHEMA public  
TO amar ;
```

GRANT all privileges on database

```
GRANT ALL  
ON DATABASE greencycles  
TO amar ;
```

# Privileges

How to grant acces?  
Typical statements

```
GRANT createdb
```

```
ALTER USER amar CREATEDB;
```

```
GRANT roles to user
```

```
GRANT sarah TO amar;
```

```
GRANT roles to user
```

```
GRANT analyst TO amar;
```

# Privileges

How to grant acces?  
Typical statements

REVOKE INSERT

```
REVOKE INSERT ON customer FROM amar;
```

REVOKE ALL PRIVILEGES

```
REVOKE ALL PRIVILEGES ON customer FROM PUBLIC ;
```

REVOKE ROLE

```
REVOKE analyst FROM amar;
```

# Privileges

- ✓ SELECT,
- ✓ INSERT,
- ✓ UPDATE,
- ✓ DELETE,
- ✓ TRUNCATE,
- ✓ USAGE,
- ✓ ALL

Privileges

```
REVOKE GRANT OPTION FOR privilege  
ON database_object  
FROM USER | ROLE | PUBLIC  
GRANTED BY USER | ROLE
```

# Privileges

- ✓ SELECT,
- ✓ INSERT,
- ✓ UPDATE,
- ✓ DELETE,
- ✓ TRUNCATE,
- ✓ USAGE,
- ✓ ALL

Privileges

```
GRANT SELECT  
ON ALL TABLES IN SCHEMA  
<schema_name>  
TO nikolai
```

# Privileges

Priveleges

USAGE

Roles

Users

- ✓ SELECT,
- ✓ INSERT,
- ✓ UPDATE,
- ✓ DELETE,
- ✓ TRUNCATE,
- ✓ USAGE

```
CREATE ROLE <user_name>
WITH LOGIN PASSWORD 'pwd123'
```

# Using indexes

Read  
operations

Write  
operations

# Using indexes

Read  
operations

Write  
operations

Understanding  
indexes

Types of  
indexes

Guidelines

Demo

# Using indexes

transaction_id [PK] integer	product_id character varying	customer_id integer	payment character varying	price numeric
1	P0494	4	visa	18.29
2	P0221	5	visa	1.49
3	P0625	5	visa	5.89
4	P0431	8	mastercard	11.59
5	P0058	5	mastercard	12.39

3, P0625, 5, visa

4, P0432, 8, mastercard

1, P0494, 4, visa

6, P0058, 5, mastercard

2, P0221, 5, visa

Data is stored without a particular order

```
SELECT  
product_id  
FROM sales  
WHERE customer_id = 5
```

Table scan

Read-inefficient

# Using indexes

transaction_id [PK] integer	product_id character varying	customer_id integer	payment character varying	price numeric
1	P0494	4	visa	18.29
2	P0221	5	visa	1.49
3	P0625	5	visa	5.89
4	P0431	8	mastercard	11.59
5	P0058	5	mastercard	12.39

1, P0494, 4, visa  
6, P0058, 5, mastercard  
3, P0625, 5, visa

2, P0221, 5, visa  
4, P0432, 8, mastercard

```
SELECT  
product_id  
FROM sales  
WHERE customer_id = 5
```

Location	Value
1	4
2	5
5	8

## Using Index

✓ Indexes help to make data reads faster!

❖ Slower data writes

❖ Additional storage

❖ B-tree Indexes

❖ Bitmap Indexes

# Using indexes

transaction_id [PK] integer	product_id character varying	customer_id integer	payment character varying	price numeric
1	P0494	4	visa	18.29
2	P0221	5	visa	1.49
3	P0625	5	visa	5.89
4	P0431	8	mastercard	11.59
5	P0058	5	mastercard	12.39

1, P0494, 4, visa  
6, P0058, 5, mastercard  
3, P0625, 5, visa

2, P0221, 5, visa  
4, P0432, 8, mastercard

```
SELECT  
product_id  
FROM sales  
WHERE customer_id = 5
```

Location	Value
1	4
2	5
5	8

# Using indexes

Location	Value
1	4
2	5
5	8

- ✓ Different types of indexes  
for different situations

❖ B-tree Indexes

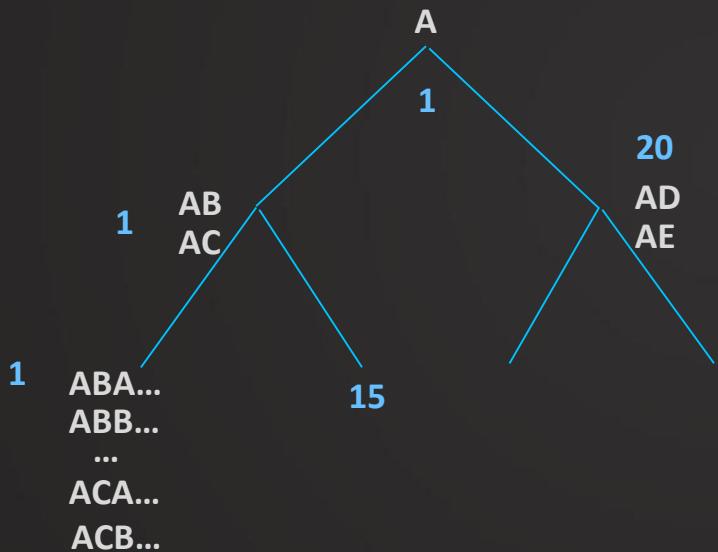
❖ Bitmap Indexes

1, P0494, 4, visa  
6, P0058, 5, mastercard  
3, P0625, 5, visa

2, P0221, 5, visa  
4, P0432, 8, mastercard



## B-tree Indexes



- ✓ Multi-level tree structure
- ✓ Breaks data down into pages or blocks
- ✓ Should be used for high-cardinality (unique) columns
- ✓ Not entire table (costy in terms of storage)

## Bitmap index

transaction_id [PK] integer	product_id character varying	customer_id integer	payment character varying	price numeric
1	P0494		4	visa 18.29
2	P0221		5	visa 1.49
3	P0625		5	visa 5.89
4	P0431		8	mastercard 11.59
5	P0058		5	mastercard 12.39

- ✓ Particularly good for dataware houses
- ✓ Large amounts of data + low-cardinality
- ✓ Very storage efficient
- ✓ More optimized for read & few DML-operations

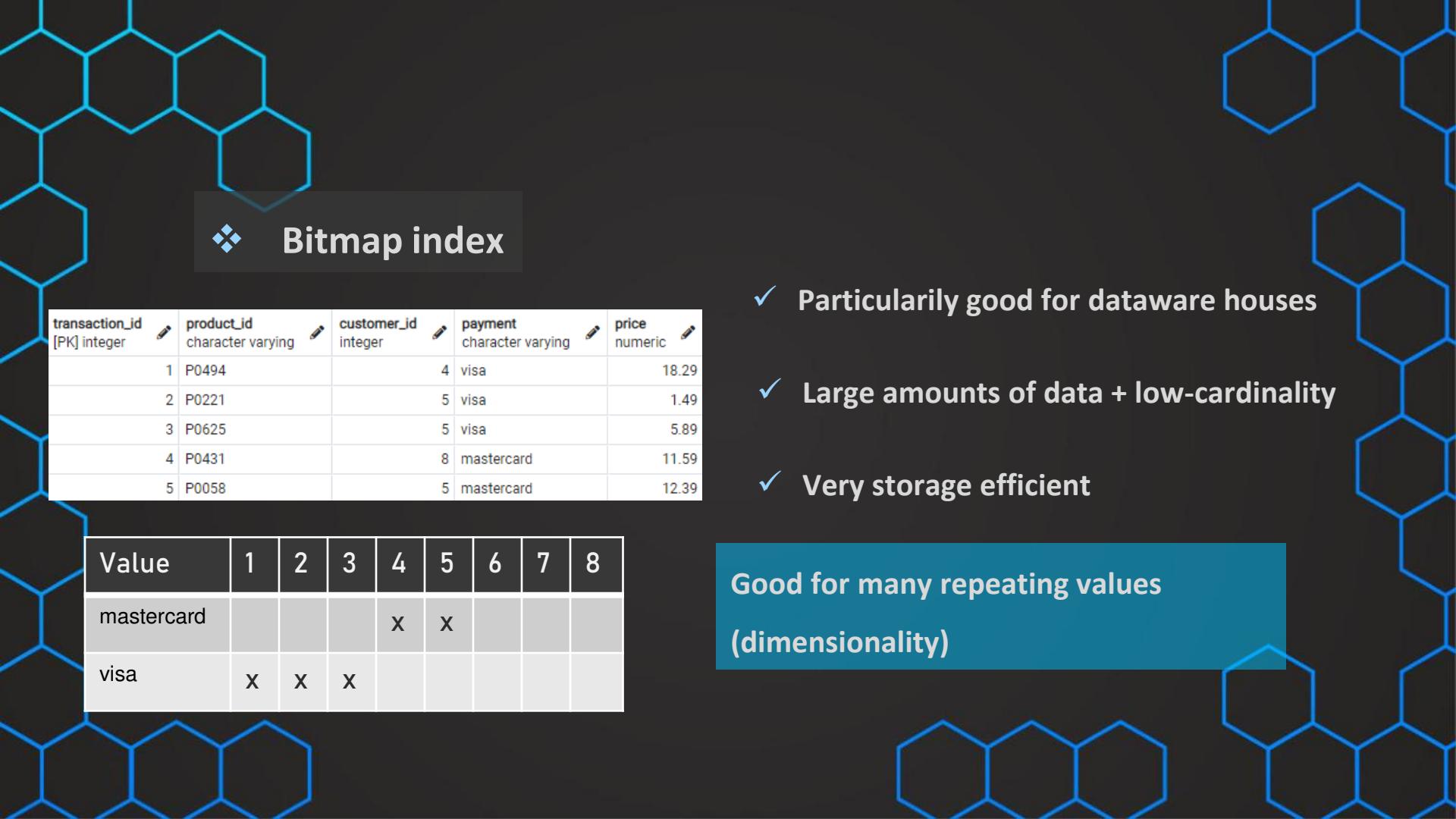
transaction_id [PK] integer	product_id character varying	customer_id integer	payment character varying	price numeric
1	P0494		4	visa
2	P0221		5	visa
3	P0625		5	visa
4	P0431		8	mastercard
5	P0058		5	mastercard

Row_id	Value	Bit
1	visa	1 1 1 0 0
4	mastercard	0 0 0 1 1

## ❖ Bitmap index

- ✓ Particularly good for dataware houses
- ✓ Large amounts of data + low-cardinality
- ✓ Very storage efficient

Good for many repeating values  
(dimensionality)



## ❖ Bitmap index

transaction_id [PK] integer	product_id character varying	customer_id integer	payment character varying	price numeric
1	P0494		4	visa
2	P0221		5	visa
3	P0625		5	visa
4	P0431		8	mastercard
5	P0058		5	mastercard

Value	1	2	3	4	5	6	7	8
mastercard				x	x			
visa	x	x	x					

- ✓ Particularly good for dataware houses
- ✓ Large amounts of data + low-cardinality
- ✓ Very storage efficient

Good for many repeating values  
(dimensionality)

# Guidelines

**B-tree Index**

**Bitmap Index**

**Default index**

**Slow to update**

**Unique columns  
(surrogate key, names)**

**Storage efficient**

**Great read performance**

# Guidelines

Should we put index on every column?

No! They come with a cost!

Storage + Create/Update time

Only when necessary!

Avoid full table reads

Small tables do not require indexes

# Guidelines

On which columns?

1. Large tables

2. Columns that are used as filters

transaction_id [PK] integer	product_id character varying	customer_id integer	payment character varying	price numeric
1	P0494	4	visa	18.29
2	P0221	5	visa	1.49
3	P0625	5	visa	5.89
4	P0431	8	mastercard	11.59
5	P0058	5	mastercard	12.39

# Guidelines

**Fact tables**

**B-tree on surrogate key**

**Bitmap key on foreign keys**

**Dimension table**

**Size of table**

**Are they used in searches a lot?**

**Choose based on cardinality**

# Demo: Creating indexes

```
CREATE INDEX index_name  
ON table_name [USING method]  
(  
    column_name  
    [, ...]  
) ;
```

# Demo: Creating indexes

```
CREATE INDEX index_name  
ON table_name  
(  
    column_name  
);
```

# Demo: Creating indexes

```
CREATE INDEX index_name  
ON table_name  
(  
    column_name1,  
    column_name2  
);
```