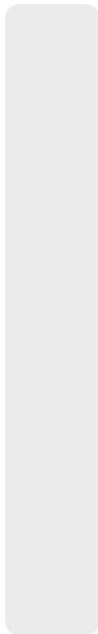


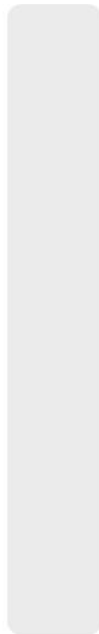
How do you feel today?

0

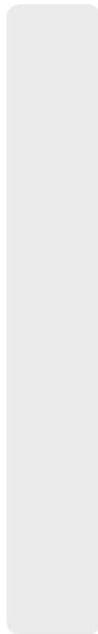
0%



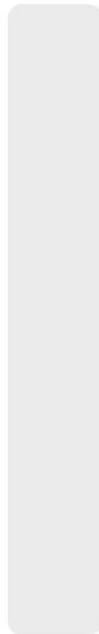
0%



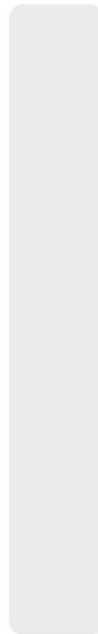
0%



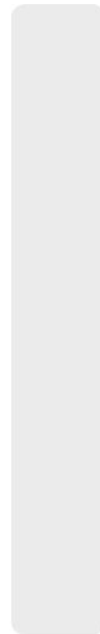
0%



0%



0%



# SQL for Exploratory Data Analysis

Associate Professor Dr. Sergio Hernandez Marin



**NUS**  
National University  
of Singapore

National University of Singapore

# SQL: An Overview

- **SQL** (Structured Query Language) is a domain-specific language used for managing and manipulating relational databases
- An Industry standard since its adoption by the American National Standards Institute (ANSI) in 1986 and the international Organization for Standardization (ISO) in 1987
- While SQL is standardized, different database management systems have their own dialects, such as T-SQL (Microsoft SQL Server), PL/SQL (Oracle), and PostgreSQL.
- SQL databases typically adhere to ACID principles, ensuring reliable transaction processing:
  - **Atomicity**: all or nothing principle
  - **Consistency**: Always in a valid state after a transaction
  - **Isolation**: Transactions operate independently of each other
  - **Durability**: Once a transaction is committed, the change made are permanent and will survive system failures

# Key Components of SQL

- **Data Definition Language (DDL):** Used to define and modify database structures
  - Examples: CREATE, ALTER, DROP, TRUNCATE
- **Data Manipulation Language (DML):** Used to manage data within database objects
  - Examples: INSERT, UPDATE, DELETE
- **Data Control Language (DCL):** Used to control access to data within the database
  - Examples: GRANT, REVOKE
- **Data Query Language (DQL):** Used to retrieve data from the database. While often considered part of DML, it's important enough to be categorized separately.
  - The primary command in DQL is SELECT.
  - DQL is the primary tool for exploratory data analysis in SQL
  - Data Retrieval, Flexibility, Performance, Integration



# SQL and Exploratory Data Analysis

Exploratory Data Analysis (EDA) is a crucial step in gaining insights from datasets before diving into complex analyses or modeling. SQL plays a powerful role in EDA with its efficient data querying and manipulation capabilities.

## How SQL Enhances EDA

- **Data Retrieval:** Use ``SELECT`` statements with clauses like ``WHERE``, ``GROUP BY``, and ``HAVING`` to extract specific subsets of data.
- **Aggregations:** Functions like ``COUNT``, ``SUM``, ``AVG``, ``MIN``, and ``MAX`` help summarize data quickly.
- **Joining Tables:** SQL's ``JOIN`` operations combine data from multiple tables, enabling more in-depth analysis.
- **Filtering:** The ``WHERE`` clause lets you focus on specific data points or ranges.
- **Sorting:** ``ORDER BY`` arranges data in meaningful orders.
- **Grouping:** ``GROUP BY`` identifies patterns and trends within categories.

# To give you a bit more of motivation...

			
Screening	<p>Project experience</p> <p><b>SQL</b></p> <p>Leadership Principle</p>	<p>Applied Stats</p> <p>Python / R</p>	<p>Python or R / <b>SQL</b></p> <p>Product Case</p>
Onsite Rounds	<p><b>SQL</b></p> <p>Business Case</p> <p>Leadership principles</p> <p>Science Breath: Data Science range and diverse knowledge</p> <p>Science Depth: Data Science expertise and understanding</p>	<p>Applied Stats</p> <p>Behavioural Interview</p> <p>Business Case</p> <p>Python or R / <b>SQL</b></p>	<p>Python or R / <b>SQL</b></p> <p>Behavioural Interview</p> <p>Analytical Reasoning</p> <p>Analytical Execution</p>



# MySQL: Origins and History

- **Founders:** MySQL was created by David Axmark, Allan Larsson, and Michael "Monty" Widenius. These pioneering developers aimed to build a flexible, open-source relational database management system (RDBMS) that could efficiently handle data for web applications.
- **Naming:** The name "MySQL" is derived from Monty Widenius's daughter, "My." This personal touch added a unique identity to the project. The dolphin logo, named Sakila, was selected through a community contest. "Sakila" is also the name of a town in Arusha, Tanzania, reflecting the global nature of MySQL's community-driven development.
- **Acquisition:** In 2008, MySQL AB, the company behind MySQL, was acquired by Sun Microsystems for approximately \$1 billion. This acquisition strengthened MySQL's position in the tech world. Subsequently, in 2010, Oracle Corporation acquired Sun Microsystems, bringing MySQL into Oracle's vast ecosystem, where it continues to be a leading open-source RDBMS.

# | What is MySQL?

**Open-Source RDBMS:** A leading choice for web applications and data management.

**Top Performance:** Celebrated for its speed, reliability, and scalability.

**Feature-Packed:** Designed to handle a wide array of tasks and requirements.

## Core Features

- **ACID Compliance:** Guarantees transaction integrity through Atomicity, Consistency, Isolation, and Durability.
- **Extensive SQL Support:** Handles all SQL types (DDL, DML, DQL, DCL).
- **Advanced Indexing:** Speeds up data retrieval with efficient indexing.
- **Stored Procedures & Functions:** Uses precompiled SQL for optimized performance.
- **Triggers:** Automates responses to data changes (INSERT, UPDATE, DELETE).
- **Replication:** Enhances data availability and load balancing across servers.

## Ideal For

- **Web Applications:** Perfect for e-commerce sites and content management systems.
- **Data Warehousing & Analytics:** Manages and retrieves large datasets efficiently.
- **Operational Databases:** Suited for high-reliability, mission-critical applications.
- **Embedded Systems:** Compact and efficient for specialized hardware



# MySQL for Exploratory Data Analysis

## Performance

- **Speed:** Renowned for fast read operations, ideal for EDA tasks.

## Scalability

- **Efficient Handling:** Manages large datasets effectively, perfect for big data exploration.

## Compatibility

- **Versatile Integration:** Works seamlessly with a variety of data analysis tools and programming languages.

## Rich Functionality

- **Built-in Functions:** Provides an extensive array of functions for data manipulation and analysis.

## Note: Differences between SQL and MySQL

	SQL	MySQL
Definition	Structured Query Language	Specific Relational Database Management System
Type	Language for managing databases	Relational Database Management System
Purpose	Interact with various DBMSs	Store, manage, and retrieve data using SQL
Standardization	Standardized language	Implementation of SQL standard
Command Types	DDL, DML, DQL, DCL	Supports DDL, DML, DQL, DCL commands
Usage	Interaction with different DBMSs	Specific choice for a DBMS
Variants	Works with various DBMSs	One specific DBMS among many
Implementation	Depends on the underlying DBMS	Open-source implementation of a DBMS
Storage engines	N/A	Supports various storage engines
Concurrency handling	N/A	Supports concurrent connections
Programming languages	Works with different programming langs	Offers client libraries for languages
Application Scope	Used in various contexts	Suitable for web apps, data warehousing, etc.
Ownership	Language standard	Developed by Oracle Corporation

# A SQL table

Used to organize and structure data in relational database management systems

Foreign key

Primary key

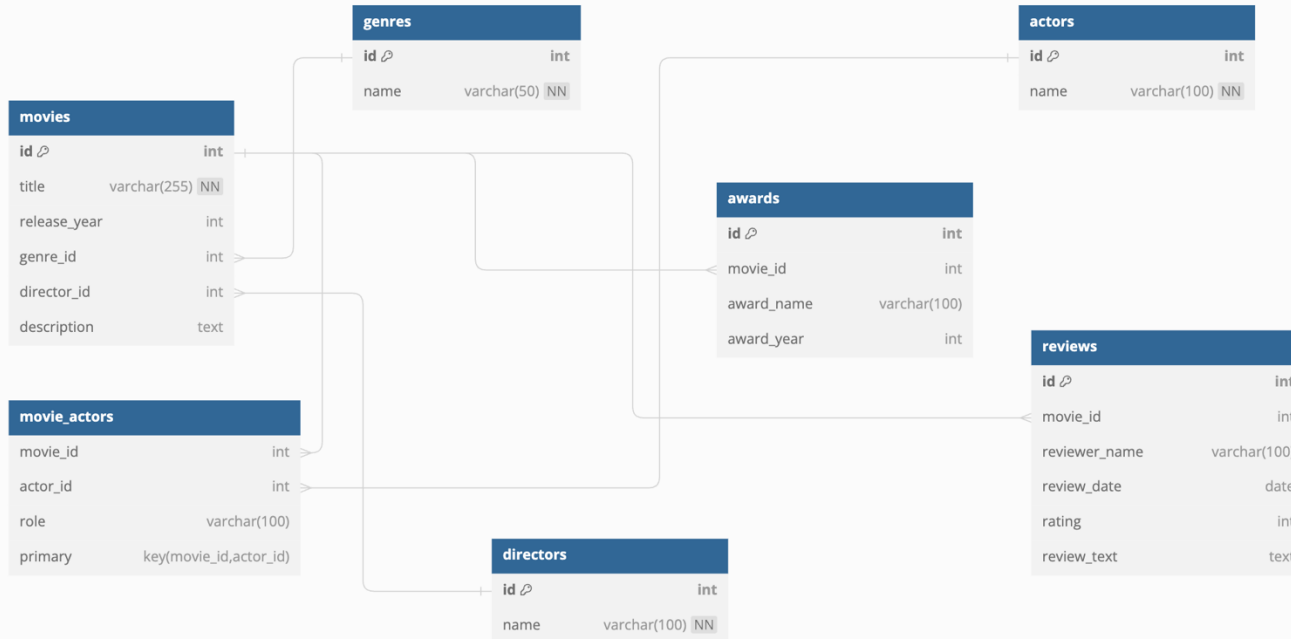
Columns (fields)

Rows (records)

customer_id	device_id	device_type	device_os	device_location	device_usage_frequency	device_ip_address	proxy_vpn_usage	device_manufacturer
1	1	Mobile Phone	iOS	Singapore, Singapore	100	10.32.79.137	0	Apple
2	2	Mobile Phone	Android	New Melanie, Canada	95	10.184.220.134	0	Samsung
3	3	Mobile Phone	Android	Singapore, Singapore	14	10.66.198.198	0	Samsung
3	4	Mobile Phone	Android	Singapore, Singapore	20	10.66.198.198	0	Samsung
3	5	Mobile Phone	Android	Singapore, Singapore	17	10.66.198.198	0	Samsung
4	6	Mobile Phone	Android	New Wasin, Thailand	111	10.202.200.37	0	Samsung
5	7	Mobile Phone	iOS	Kevin Ville, Philippines	35	192.168.183.145	0	Apple
5	8	Tablet	iOS	Kevin Ville, Philippines	32	192.168.183.145	0	Apple
5	9	Mobile Phone	iOS	Kevin Ville, Philippines	31	192.168.183.145	0	Apple
6	10	Mobile Phone	iOS	Judyhaven, Canada	76	192.168.248.86	0	Apple
7	11	Mobile Phone	iOS	Todd Ville, Philippines	66	172.28.162.100	0	Apple
7	12	Mobile Phone	iOS	Todd Ville, Philippines	55	172.28.162.100	0	Apple
8	13	Mobile Phone	Android	Staciefurt, Australia	80	10.192.48.108	0	Samsung
9	14	Mobile Phone	iOS	Uluberia, India	34	172.24.140.93	0	Apple
9	15	Desktop	iOS	Uluberia, India	43	172.24.140.93	0	Apple
10	16	Mobile Phone	Android	Austinview, Canada	31	192.168.247.18	0	Samsung
10	17	Tablet	Android	Austinview, Canada	22	192.168.247.18	0	Samsung
10	18	Desktop	Windows	Austinview, Canada	23	192.168.247.18	0	Samsung

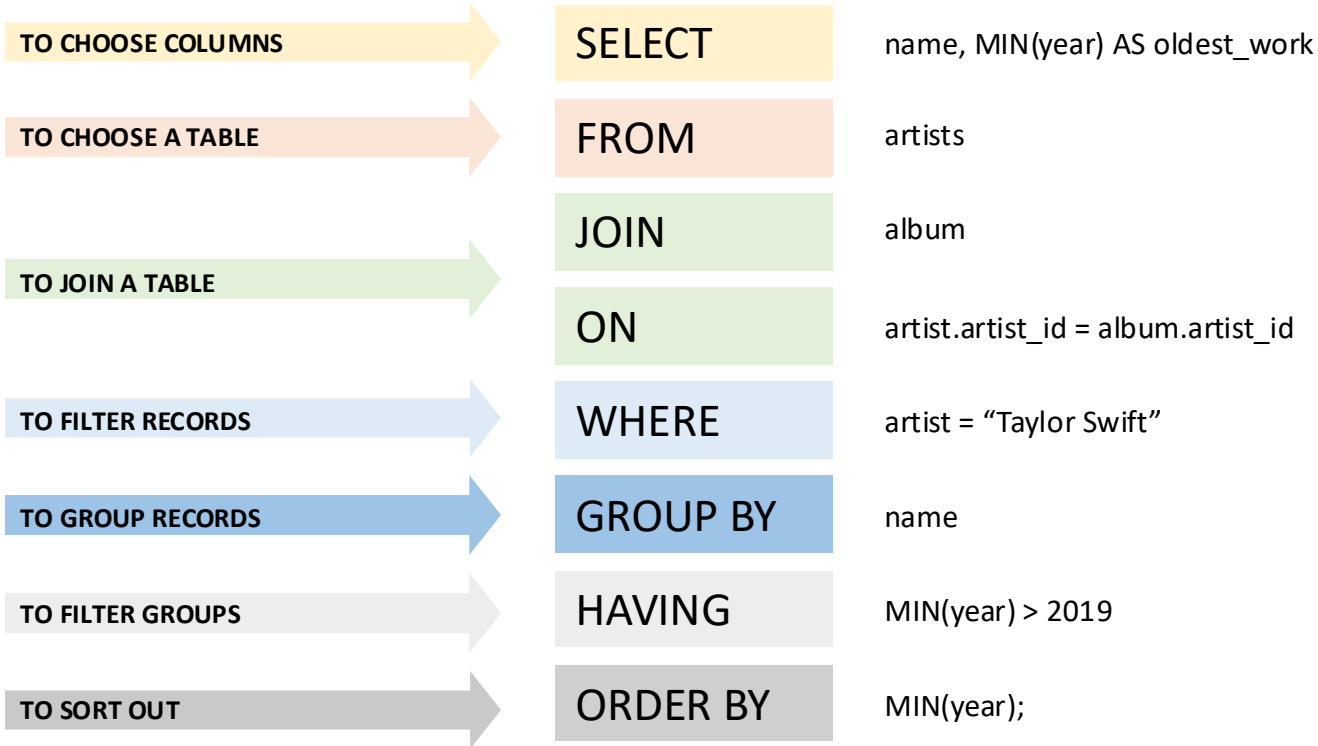
# Entity-Relationship Diagram

A visual representation used to model the relationships between different entities (objects or concepts) in a database system



# Understanding SQL Queries

## BASIC STRUCTURE



## Query Example

### Example

```
SELECT
  title AS movie_title,
  release_year,
FROM
  Movie
WHERE
  release_year >= desired_start_year
  AND release_year <= desired_end_year
ORDER BY
  title ASC
LIMIT 10;
```

- **SELECT** statement to specify the columns to retrieve.
- **FROM** clause to specify the source table.
- **WHERE** clause to filter data based on specific conditions to retrieve only the relevant information
- **ORDER BY** clause to sort and order (ASC / DESC)
- **LIMIT** clause: Controlling the number of rows returned by the query
  - Useful for quick data sampling and initial exploration
- The **AS** keyword is used to provide an alias for a table or a column. An alias is an alternative name used to make:
  - Things more readable
  - To avoid name conflicts

# Basic SQL Syntax

Keyword	Meaning	Keyword	Meaning
<b>SELECT</b>	Retrieves data from a table	<b>INSERT INTO</b>	Adds new data to a table
<b>FROM</b>	Specifies the table(s) to retrieve data from	<b>UPDATE</b>	Modifies existing data in a table
<b>WHERE</b>	Filters data based on conditions	<b>DELETE</b>	Deletes data from a table
<b>GROUP BY</b>	Groups data by one or more columns	<b>CREATE TABLE</b>	Creates a new table in the database
<b>HAVING</b>	Filters grouped data based on conditions	<b>ALTER TABLE</b>	Modifies an existing table (e.g., adding or dropping columns)
<b>ORDER BY</b>	Sorts data in ascending or descending order	<b>DROP TABLE</b>	Deletes a table from the database
<b>JOIN</b>	Combines rows from two or more tables	<b>INNER JOIN</b>	Returns rows with at least one match in both tables
<b>LEFT JOIN</b>	Returns all rows from the left table	<b>RIGHT JOIN</b>	Returns all rows from the right table
<b>FULL OUTER JOIN</b>	Returns all rows when there is a match in either table	<b>DISTINCT</b>	Retrieves unique values in a column
<b>AS</b>	Renames a column or table with an alias	<b>LIKE</b>	Searches for a specified pattern using wildcard characters
<b>IN</b>	Specifies multiple values for a column	<b>BETWEEN</b>	Selects values within a given range
<b>NULL</b>	Represents NULL values in the database	<b>IS NULL</b>	Checks if a value is NULL
<b>NOT NULL</b>	Checks if a value is not NULL		

# Basic SQL Syntax

## **SELECT**

Allows users to retrieve data from one or more tables within a database. It follows this basic structure:

```
SELECT column1, column2, ...  
FROM table_name;
```

For example:

```
SELECT * FROM customers;
```

This retrieves all data from the "customers" table.

## **FROM**

The FROM clause specifies the table(s) to retrieve data from.

```
SELECT column1, column2, ...  
FROM tablename1, tablename2, ...;
```

For example:

```
SELECT order_id FROM orders
```

This retrieves data from the "orders" and "customers" tables where the customer ID matches.



## Basic SQL Syntax

### WHERE

Used to filter records based on specific conditions. The WHERE clause is typically used with the SELECT statement to filter the results and retrieve only the rows that meet specific conditions.

```
SELECT column1, column2, ...  
FROM table_name  
WHERE condition;
```

For example:

```
SELECT * FROM orders WHERE order_date >= '2024-01-01';
```

This retrieves data from the "orders" table where the "order\_date" is "2024-01-01" or greater than "2024-01-01"

### FROM

Used to sort the result set in ascending or descending order based on one or more columns.

```
SELECT column1, column2, ...  
FROM table_name  
ORDER BY column1 [ASC|DESC];
```

For example:

```
SELECT * FROM products ORDER BY price ASC;
```

This retrieves data from the "orders" and "customers" tables where the customer ID matches.

## Basic SQL Syntax

### GROUP BY

The GROUP BY clause is used to group rows that have the same values into summary rows, typically used with aggregate functions like COUNT, SUM, AVG, etc.

```
SELECT column1, aggregate_function(column2)
FROM table_name
GROUP BY column1;
```

For example:

```
SELECT customer_id, COUNT(order_id) FROM orders GROUP BY customer_id;
```

This groups data from the "orders" table by customer ID and counts the number of orders for each customer.

### HAVING

Used to filter records returned by the GROUP BY clause based on specified conditions.

```
SELECT column1, aggregate_function(column2)
FROM table_name
GROUP BY column1
HAVING condition;
```

#### EXAMPLE

```
SELECT customer_id, COUNT(order_id)
FROM orders
GROUP BY customer_id
HAVING COUNT(order_id) > 5;
```

This groups data from the "orders" table by customer ID, counts the number of orders for each customer, and filters the results to only show customers with more than 5 orders.

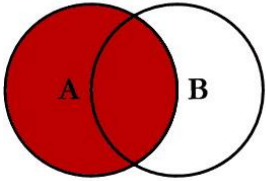
# Unity is strength: JOINS

## Example

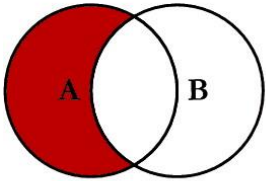
```
SELECT
  e.FirstName,
  e.LastName,
  d.DepartmentName
FROM
  Employees AS e
INNER JOIN
  Departments AS d
ON Employees.DepartmentID = Departments.DepartmentID;
```

- Combine data from two or more tables based on a related column between them
- Primary keys uniquely ID rows, while foreign keys connect to the primary key in another table. Joins use these connections to match keys.
- The **ON** clause specifies the conditions for joining tables.
- Consider how you handle **NULL** values
- The order in which tables are listed in the query can sometimes affect the results
- Careful with joins, they can impact performance

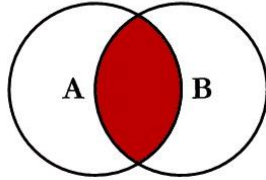
# SQL JOINS



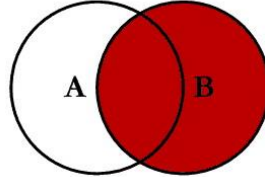
```
SELECT <select_list>  
FROM TableA A  
LEFT JOIN TableB B  
ON A.Key = B.Key
```



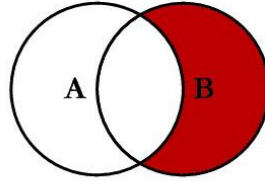
```
SELECT <select_list>  
FROM TableA A  
LEFT JOIN TableB B  
ON A.Key = B.Key  
WHERE B.Key IS NULL
```



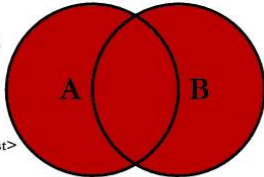
```
SELECT <select_list>  
FROM TableA A  
INNER JOIN TableB B  
ON A.Key = B.Key
```



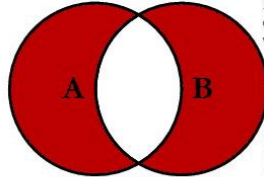
```
SELECT <select_list>  
FROM TableA A  
RIGHT JOIN TableB B  
ON A.Key = B.Key
```



```
SELECT <select_list>  
FROM TableA A  
RIGHT JOIN TableB B  
ON A.Key = B.Key  
WHERE A.Key IS NULL
```



```
SELECT <select_list>  
FROM TableA A  
FULL OUTER JOIN TableB B  
ON A.Key = B.Key
```



```
SELECT <select_list>  
FROM TableA A  
FULL OUTER JOIN TableB B  
ON A.Key = B.Key  
WHERE A.Key IS NULL  
OR B.Key IS NULL
```

© C.L. Moffatt, 2008

## Combining data

- Typically, multiple tables are used to organize data efficiently while minimizing data redundancy
- Benefits:
  - Data Organization
  - Normalization
  - Efficient storage
  - Scalability
  - Flexibility
  - Data Integrity
  - Performance
  - Security
  - Analytical capabilities
  - Maintainability

## A Join Example

```
SELECT
  e.FirstName,
  e.LastName,
  d.DepartmentName
FROM
  Employees AS e
INNER JOIN
  Departments AS d
ON Employees.DepartmentID = Departments.DepartmentID;
```

- Combine data from two or more tables based on a related column between them
- Primary keys uniquely ID rows, while foreign keys connect to the primary key in another table. Joins use these connections to match keys.
- The **ON** clause specifies the conditions for joining tables.
- Consider how you handle **NULL** values
- The order in which tables are listed in the query can sometimes affect the results
- Careful with joins, they can impact performance

# Inner Join

What if you want to have a table that contains only users that have done an action?

user table

id (Primary Key)	Name	Address
1	John Doe	123 Main St
2	Jane Smith	456 Oak Ave
3	Michael Johnson	789 Elm Blvd

events table

user_id (Foreign Key)	id (Primary Key)	Action
1	1	Viewed profile
3	2	Sent message
4	3	Liked post

Two tables are linked by a common column: User ID. This column is the primary key in the User Table and a foreign key in the Event Table. By joining these tables on this column, we can combine user information with event data.

An inner join combines data from two tables based on a shared column. Only rows with matching values in this column will be included in the result.

```
SELECT
  u.id, u.name, u.address,
  e.action
FROM
  users AS u
INNER JOIN
  events AS e
ON u.id = e.user_id;
```

id	Name	Address	Action
1	John Doe	123 Main St	Viewed profile

## Left Join

what if you want to have a table that contains all the users' data and only actions that those users have done? Actions performed by other users not in the users table should not be included?

user table

id (Primary Key)	Name	Address
1	John Doe	123 Main St
2	Jane Smith	456 Oak Ave
3	Michael Johnson	789 Elm Blvd

events table

user_id (Foreign Key)	id (Primary Key)	Action
1	1	Viewed profile
3	2	Sent message
4	3	Liked post

Two tables are linked by a common column: User ID. This column is the primary key in the User Table and a foreign key in the Event Table. By joining these tables on this column, we can combine user information with event data.

A left join shows all data from the first table, even if there's no matching data in the second table. It combines matching rows based on a shared column

```
SELECT
  u.id, u.name, u.address,
  e.action
FROM
  users AS u
LEFT JOIN
  events AS e
ON u.id = e.user_id;
```

id	Name	Address	Action
1	John Doe	123 Main St	Viewed profile
2	Jane Smith	456 Oak Ave	

## Cross Join

user table

id (Primary Key)	Name	Address
1	John Doe	123 Main St
2	Jane Smith	456 Oak Ave
3	Michael Johnson	789 Elm Blvd

events table

user_id (Foreign Key)	id (Primary Key)	Action
1	1	Viewed profile
3	2	Sent message
4	3	Liked post

Two tables are linked by a common column: User ID. This column is the primary key in the User Table and a foreign key in the Event Table. By joining these tables on this column, we can combine user information with event data.

A **Cross Join** would result in a table with all possible combinations of your tables' rows together. This can result in enormous tables and should be used with caution.

```
SELECT
  u.id, u.name, u.address,
  e.action
FROM
  users AS u
CROSS JOIN
  events AS e;
```

Name	Address	Action
John Doe	123 Main St	Viewed profile
John Doe	123 Main St	Sent message
John Doe	123 Main St	Liked post
Jane Smith	456 Oak Ave	Viewed profile
Jane Smith	456 Oak Ave	Sent message
Jane Smith	456 Oak Ave	Liked post
Michael Johnson	789 Elm Blvd	Viewed profile
Michael Johnson	789 Elm Blvd	Sent message
Michael Johnson	789 Elm Blvd	Liked post



# Union

user table

id (Primary Key)	Name	Address
1	John Doe	123 Main St
2	Jane Smith	456 Oak Ave
3	Michael Johnson	789 Elm Blvd

former\_users table

id (Primary Key)	Name	Address
4	Sarah Connor	321 Pine St
5	Kyle Reese	654 Maple Ave

A **Union Join** will stack tables on top of each other resulting in new rows.

```
SELECT
  id, name, address
FROM
  users
UNION
SELECT
  id, name, address
FROM
  former_users;
```

id	Name	Address
1	John Doe	123 Main St
2	Jane Smith	456 Oak Ave
3	Michael Johnson	789 Elm Blvd
4	Sarah Connor	321 Pine St
5	Kyle Reese	654 Maple Ave

---

# Hands-on exercises

---

## Exercise 1: Create your own database

1. Create a database Music database
2. It will have 3 tables:
  - artists with fields: artistid, name, genre
  - albums with fields: albumid, title, releasedate, artistid
  - tracks with fields: trackid, title, duration, albumid
3. Note on relationships:
  - An artist can have many albums (one-to-many)
  - An album is associated with one artist (many-to-one)
  - An album can have many tracks (one-to-many)

# Exercise 1 (Solution): Create your own database

Create database and tables

## MySQL Code

1. CREATE DATABASE music;
2. USE music;

```
CREATE TABLE artists (  
  artistid INT PRIMARY KEY,  
  name VARCHAR(255),  
  genre VARCHAR(255)  
);
```

```
CREATE TABLE albums (  
  albumid INT PRIMARY KEY,  
  title VARCHAR(255),  
  releasedate DATE,  
  artistid INT,  
  FOREIGN KEY (artistid) REFERENCES artists(artistid)  
);
```

```
CREATE TABLE tracks (  
  trackid INT PRIMARY KEY,  
  title VARCHAR(255),  
  duration INT,  
  albumid INT,  
  FOREIGN KEY (albumid) REFERENCES albums(albumid)  
);
```

## Exercise 1 (Solution): Create your own database

Add some data

```
INSERT INTO artists (artistid, name, genre)
VALUES
(1, 'Taylor Swift', 'Pop'),
(2, 'Beyoncé', 'R&B'),
(3, 'Shia LaBeouf', 'Hip-Hop');
```

```
INSERT INTO albums (albumid, title, releasedate, artistid)
VALUES
(1, 'Folklore', '2020-07-23', 1),
(2, 'Lemonade', '2016-04-23', 2),
(3, 'The Campaign', '2020-09-19', 3);
```

```
INSERT INTO tracks (trackid, title, duration, albumid)
VALUES
(1, 'Cardigan', 240, 1),
(2, 'Formation', 270, 2),
(3, 'Honey Boy', 300, 3);
```

## Exercise 2: The Basics

Using the Sakila database

1. Basic customer data exploration (customer table)
  - a) Display the first name, last name, and email of customers who are active. Sort the result by last name in alphabetical order
  - b) Count the number of active and inactive customers in the database. Display the active status and the corresponding count.
2. Basic Film Data Exploration (film table)
  - a) List all films that have a 'PG-13' rating. Display the film title and rating.
  - b) Count the number of films available for each rating. Display the rating and the corresponding film count, ordered by film count in descending order.

## Exercise 2 (Solutions): The basics

Basic customer data exploration

### 1.a

```
SELECT
  first_name,
  last_name,
  email
FROM
  customer
WHERE
  active = 1
ORDER BY
  last_name ASC;
```

### 1.b

```
SELECT
  active,
  COUNT(*) AS customer_count
FROM
  customer
GROUP BY
  active;
```

## Exercise 2 (Solutions): The basics

Basic film data exploration

### 2.a

```
SELECT
  title,
  rating
FROM
  film
WHERE
  rating = 'PG-13';
```

### 2.b

```
SELECT
  rating,
  COUNT(*) AS film_count
FROM
  film
GROUP BY
  rating
ORDER BY
  film_count DESC;
```



## Exercise 3: Joins

Using the Sakila database

1. List the first name, last name, and payment amount for each customer who has made a payment. (customer and payment tables)
2. Display the first name, last name, and payment amount for all staff members, even those who haven't made any payments. (staff and payment tables)
3. Create all possible combinations of actor names and film categories.

## Exercise 3 (Solutions): Joins

Using the Sakila Database

1

```
SELECT
  c.first_name,
  c.last_name,
  p.amount
FROM
  customer AS c
INNER JOIN
  payment AS p
  ON c.customer_id = p.customer_id
LIMIT
  10;
```

2

```
SELECT
  st.first_name,
  st.last_name,
  p.amount
FROM
  staff AS st
LEFT JOIN
  payment AS p
  ON st.staff_id = p.staff_id
LIMIT
  10;
```

3

```
SELECT
  a.first_name,
  a.last_name,
  c.name AS category
FROM
  actor a
CROSS JOIN
  category c
LIMIT
  15;
```

## Exercise 4: Advanced SQL queries

Using the Sakila database

1. Subqueries
  - a) Write a query to list the titles and lengths of films that are shorter than the average film length
2. Common Table Expression
  - a) Find Top 10 Most Expensive Films by Rental Rate Use a CTE to calculate and list the top 10 films by rental rate

## Exercise 3 (Solutions): Advanced SQL queries

Using the Sakila Database

### 1.a

```
SELECT
    title,
    length
FROM
    film
WHERE
    length < (
        SELECT
            AVG(length)
        FROM
            film
    );
```

### 2.a

```
WITH top_rental_rates AS (
    SELECT
        title,
        rental_rate
    FROM
        film
    ORDER BY
        rental_rate DESC
    LIMIT
        10
)
SELECT
    title,
    rental_rate
FROM
    top_rental_rates;
```

Nobody has responded yet.

Hang tight! Responses are coming in.

---

# Optional Session

---

# AirBnB Case Study; Hands-On Session

# **That's All for Today**