SESSIONS 1

Introduction to Databases

Data Science Program

Purwadhika
Startup and Coding School

Outline

- What is databases?
- Why we need databases?

Purwadhika
Startup and Coding School

What is Databases?

Purwadhika
Startup and Coding School

Database Definition

- A **database** is an organized collection of data, generally stored and accessed electronically from a computer system.
- **Database**, also called electronic database, any collection of data, or information, that is specially organized for rapid search and retrieval by a computer.
- Where databases are more complex they are often developed using formal design and modeling techniques.

Purwadhika
Startup and Coding School

Database Terminology

- Formally, a "database" refers to a set of related data and the way it is organized.
- Access to this data is usually provided by a "database management system" (DBMS) consisting of an integrated set of computer software that allows users to interact with one or more databases and provides access to all of the data contained in the database (although restrictions may exist that limit access to particular data).

Why we need database?

Purwadhika
Startup and Coding School

Database Functionality

1. Database Management System (DBMS) help to correlate, query, and report to the collected information.
2. DBMS can quickly answer lots of questions.
3. DBMS help to understand complicated data.
4. DBMS can improve our work and reducing excessive work hours in pursuit of particular objectives.

Purwadhika
Startup and Coding School

Using Databases to Improve Business Performance and Decision-Making

Businesses use their databases to:

1. Keep track of basic transaction
2. Provide information that will help the company run the business more efficiently
3. Help managers and employees make better decisions

Purwadhika
Startup and Coding School

Reference

- Wikipedia, “Database”, https://en.wikipedia.org/wiki/Database
- Britannica, “Database”, https://www.britannica.com/technology/database
- David Scott Brown, “7 Reasons Why You Need a Database Management System”, https://www.techopedia.com/2/31970/it-business/7-reasons-why-you-need-a-database-management-system
- Using Databases to Improve Business Performance and Decision-Making, https://paginas.fe.up.pt/~als/mis10e/ch6/chpt6-3bullettext.htm

Purwadhika
Startup and Coding School

SESSIONS 1

CREATE Table Statement

Data Science Program

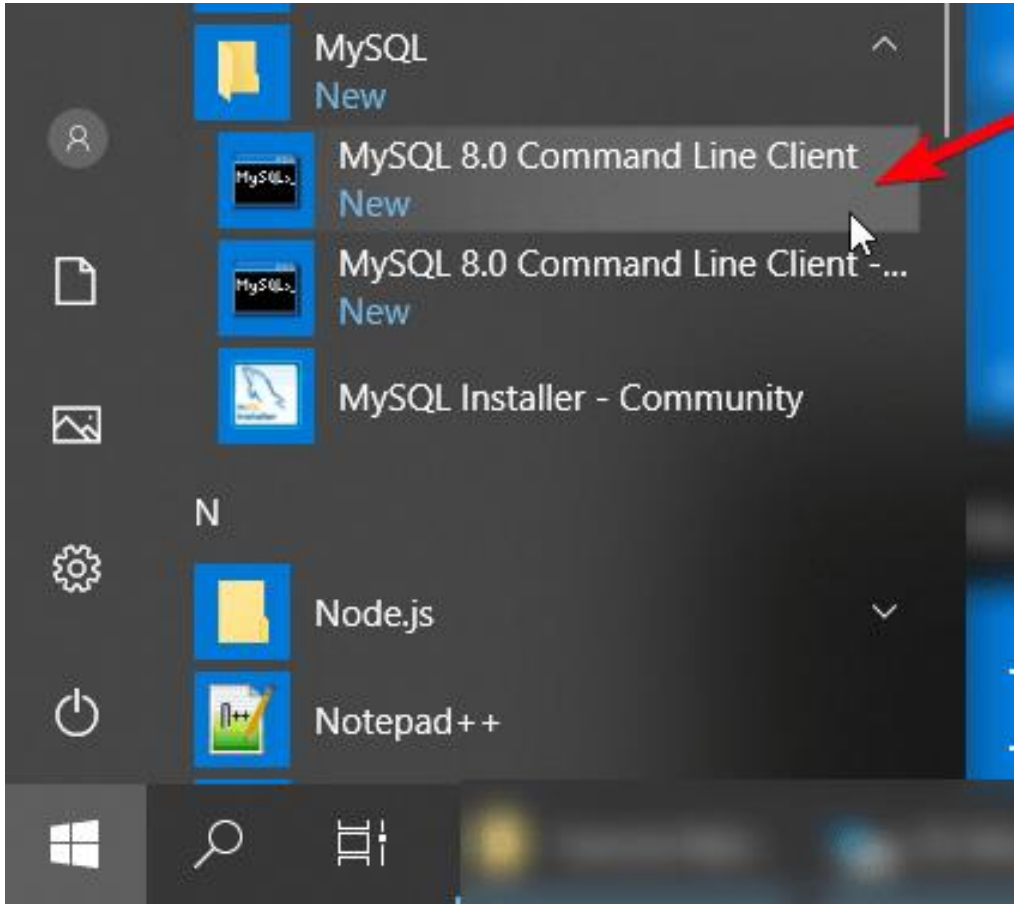
Purwadhika
Startup and Coding School

How to Access Databases in MySQL?

MySQL Databases

Purwadhika
Startup and Coding School

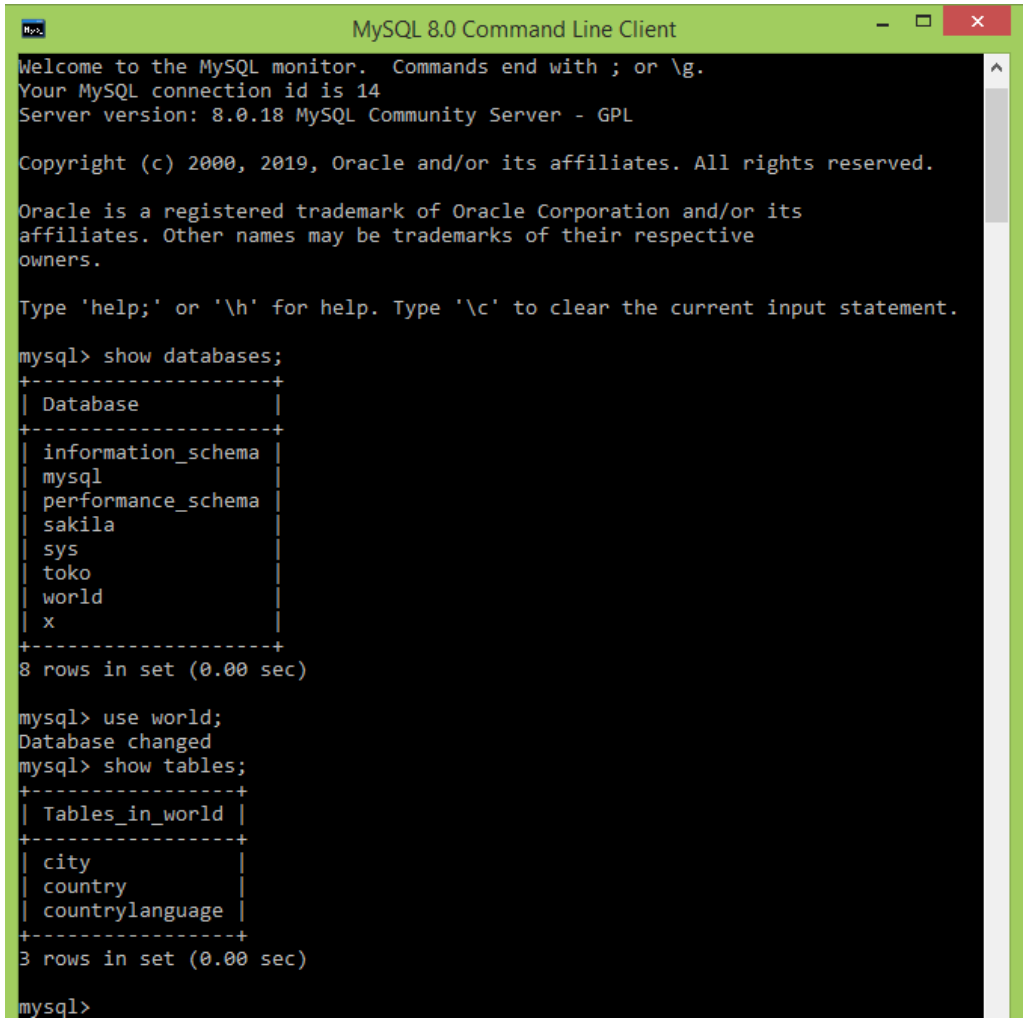
Access Databases using **MySQL 8.0 Command Line Client**



- Click Windows logo
- Search **MySQL 8.0 Command Line Client**

Purwadhika
Startup and Coding School

Access Databases using **MySQL 8.0 Command Line Client**



- Type your **password.**
- If you see *"**mysql>**"*, it means you have successfully enter access databases.
- Type '***show databases;***' to view all databases.
- Type '***use databasename***' to access one of your databases. For example: *use world*.
- To view all tables in your database, type *"**show tables**'*

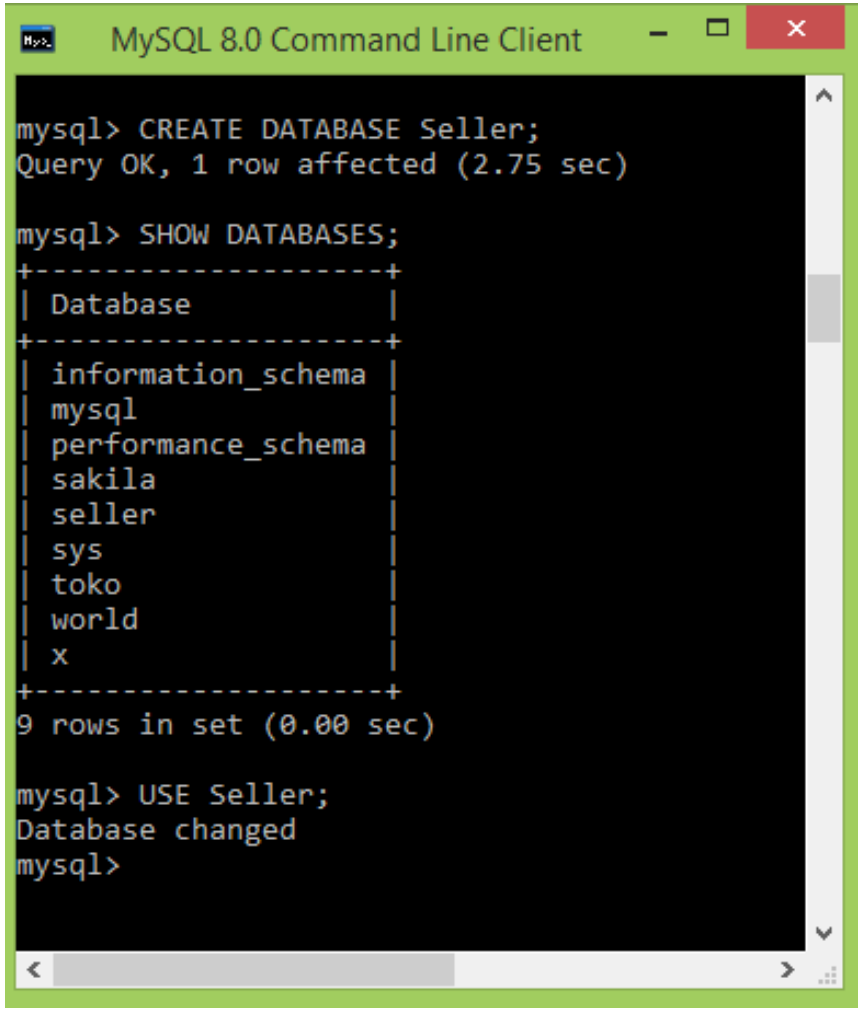
Purwadhika
Startup and Coding School

Create & Drop Database

MySQL Databases

Purwadhika
Startup and Coding School

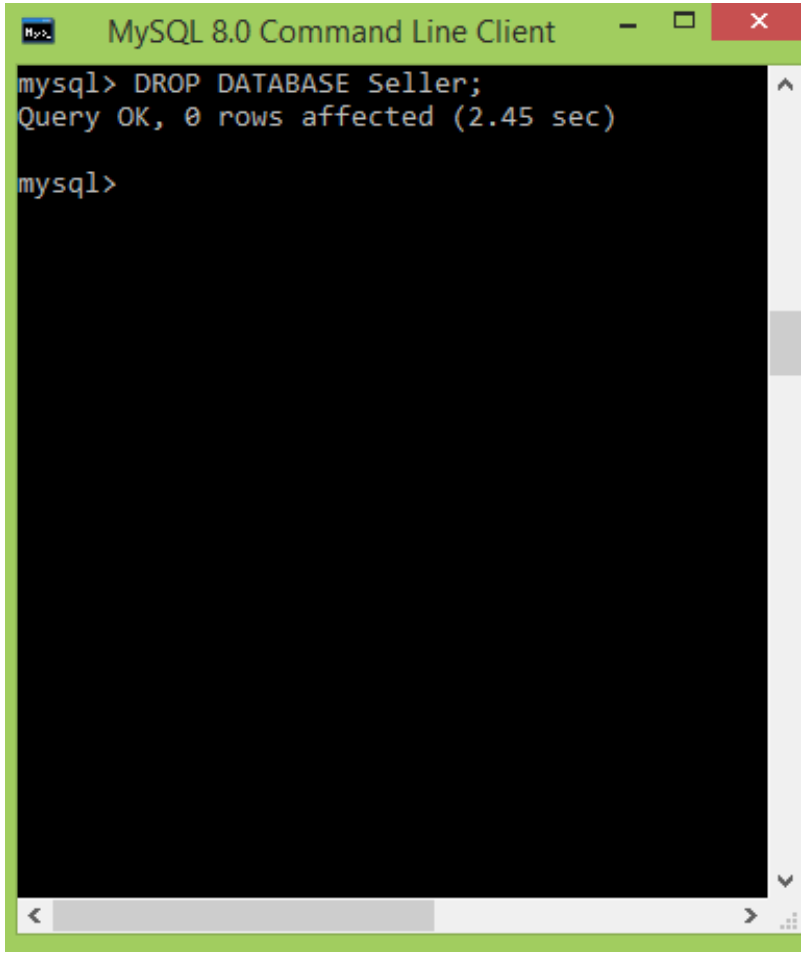
Create Database



- To create new **databases**, type: CREATE DATABASE database_name;
- For example: **CREATE DATABASE Seller;**
- If you want to use this database, type **USE Seller;**

Purwadhika
Startup and Coding School

Drop Database



- Sometimes, we no longer using database. We can drop this database, just type: **DROP DATABASE *databasename*;**
- For example, type: **DROP DATABASE Seller;**
- But if you still need this database and tables inside database, don't do this syntax!

Purwadhika
Startup and Coding School

Create Table

MySQL Databases

Purwadhika
Startup and Coding School

CREATE TABLE Statement

- The CREATE TABLE statement is used to create a new table in a database.

```
CREATE TABLE table_name (
    column1 datatype,
    column2 datatype,
    column3 datatype,
   ....
);
```

- The column parameters specify the names of the columns of the table.
- The datatype parameter specifies the type of data the column can hold (e.g. varchar, integer, date, etc.).

Purwadhika
Startup and Coding School

Example

- The following example creates a table called "Persons" that contains five columns: PersonID, LastName, FirstName, Address, and City:

```
CREATE TABLE Persons (
    PersonID int,
    LastName varchar(255),
    FirstName varchar(255),
    Address varchar(255),
    City varchar(255)
);
```

- The PersonID column is of type int and will hold an integer.
- The LastName, FirstName, Address, and City columns are of type varchar and will hold characters, and the maximum length for these fields is 255 characters.

Purwadhika
Startup and Coding School

Create Table Using Another Table

- A copy of an existing table can also be created using CREATE TABLE.
- The new table gets the same column definitions. All columns or specific columns can be selected.
- If you create a new table using an existing table, the new table will be filled with the existing values from the old table.

```
CREATE TABLE new_table_name AS
    SELECT column1, column2,...
    FROM existing_table_name
    WHERE ....;
```

Purwadhika
Startup and Coding School

Example

- The following SQL creates a new table called "TestTables" (which is a copy of the "Customers" table):

```
CREATE TABLE TestTable AS
SELECT customername, contactname
FROM customers;
```

Reference

- https://www.w3schools.com/sql/sql_create_db.asp
- https://www.w3schools.com/sql/sql_drop_db.asphttps://www.w3schools.com/sql/sql_drop_db.asp
- https://www.w3schools.com/sql/sql_create_table.asp

Purwadhika
Startup and Coding School

SESSIONS 1

SELECT Table Statement

Data Science Program

Purwadhika
Startup and Coding School

SELECT TABLE Statement

- The SELECT statement is used to select data from a database.
- The data returned is stored in a result table, called the result-set.

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

- Here, column1, column2, ... are the field names of the table you want to select data from. If you want to select all the fields available in the table, use the following syntax:

```
SELECT * FROM table_name;
```

Purwadhika
Startup and Coding School

Example

- The following example select all column in a table called “City" that contains five columns: ID, Name, CountryCode, District, and Population:

SELECT * FROM CITY;

ID	Name	CountryCode	District	Population
1	Kabul	AFG	Kabol	1780000
2	Qandahar	AFG	Qandahar	237500
3	Herat	AFG	Herat	186800
4	Mazar-e-Sharif	AFG	Balkh	127800
5	Amsterdam	NLD	Noord-Holland	731200

Example

- Select only column ‘Name’, ‘District’, and ‘Population’ from “City" Table:

SELECT Name, District, Population FROM City;

Name	District	Population
Kabul	Kabol	1780000
Qandahar	Qandahar	237500
Herat	Herat	186800
Mazar-e-Sharif	Balkh	127800
Amsterdam	Noord-Holland	731200

Purwadhika
Startup and Coding School

Reference

- https://www.w3schools.com/sql/sql_select.asp

Purwadhika
Startup and Coding School

SESSIONS 1

LIMIT, DISTINCT, COUNT, AVG, SUM

Data Science Program

Purwadhika
Startup and Coding School

SELECT DISTINCT

Purwadhika
Startup and Coding School

SELECT DISTINCT

- The SELECT DISTINCT statement is used to return only distinct (different) values.
- Inside a table, a column often contains many duplicate values; and sometimes you only want to list the different (distinct) values.

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

SELECT DISTINCT

- The following SQL statement selects only the DISTINCT values from the “**Name**" column in the “City" table:

```
SELECT DISTINCT Name
FROM City;
```

```
+-----------------+
| name            |
+-----------------+
| Kabul           |
| Qandahar        |
| Herat           |
| Mazar-e-Sharif  |
| Amsterdam       |
+-----------------+
```

Purwadhika
Startup and Coding School

SELECT DISTINCT

- The following SQL statement selects only the DISTINCT values from the “**District**" column in the “City" table:

```
SELECT DISTINCT District
FROM City;
```

```
+---------------+
| District      |
+---------------+
| Kabol         |
| Qandahar      |
| Herat         |
| Balkh         |
| Noord-Holland |
+---------------+
```

Purwadhika
Startup and Coding School

LIMIT

Purwadhika
Startup and Coding School

LIMIT

- The following SQL statement selects the first **three** records from the “City" table

```
SELECT * FROM City
LIMIT 3;
```

ID	Name	CountryCode	District	Population
1	Kabul	AFG	Kabol	1780000
2	Qandahar	AFG	Qandahar	237500
3	Herat	AFG	Herat	186800

Purwadhika
Startup and Coding School

LIMIT

- The following SQL statement selects the first **five** records from the “City" table

```
SELECT * FROM City
LIMIT 5;
```

ID	Name	CountryCode	District	Population
1	Kabul	AFG	Kabol	1780000
2	Qandahar	AFG	Qandahar	237500
3	Herat	AFG	Herat	186800
4	Mazar-e-Sharif	AFG	Balkh	127800
5	Amsterdam	NLD	Noord-Holland	731200

COUNT

Purwadhika
Startup and Coding School

COUNT

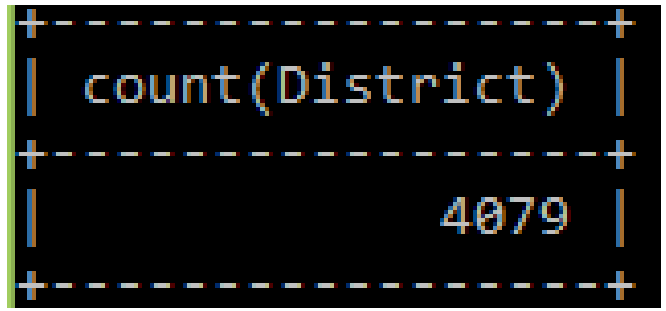
- The COUNT() function returns the number of rows that matches a specified criterion.

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

COUNT

- The following SQL statement finds the number of **District** from City table:

```
SELECT COUNT(District)
FROM City;
```



```
+-----------------+
| count(District) |
+-----------------+
|            4079 |
+-----------------+
```

Purwadhika
Startup and Coding School

COUNT

- The following SQL statement finds the number of **Continent** from Country table:

```
SELECT COUNT(Continent)
FROM Country;
```

```
+--------------------+
| COUNT(Continent)   |
+--------------------+
|                239 |
+--------------------+
```

Purwadhika
Startup and Coding School

AVG

Purwadhika
Startup and Coding School

AVG

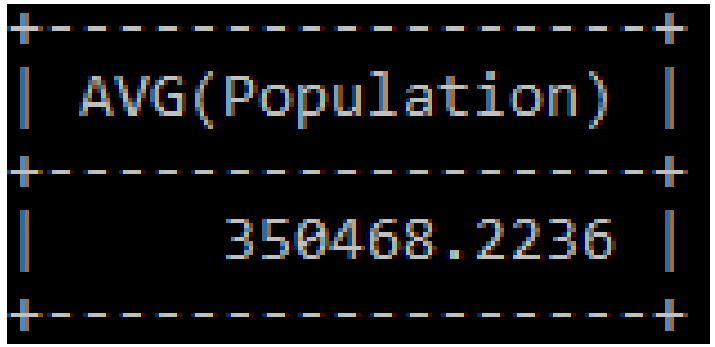
- The AVG() function returns the average value of a numeric column.

```
SELECT AVG(column_name)
FROM table_name;
```

AVG

- The following SQL statement finds the average **population** from city table:

```
SELECT AVG(Population)
FROM City;
```



Purwadhika
Startup and Coding School

AVG

- The following SQL statement finds the average **Life Expectancy** from country table:

```
SELECT AVG(LifeExpectancy)
FROM Country;
```

```
+-----------------------+
| AVG(LifeExpectancy)   |
+-----------------------+
|              66.48604 |
+-----------------------+
```

Purwadhika
Startup and Coding School

SUM

Purwadhika
Startup and Coding School

SUM

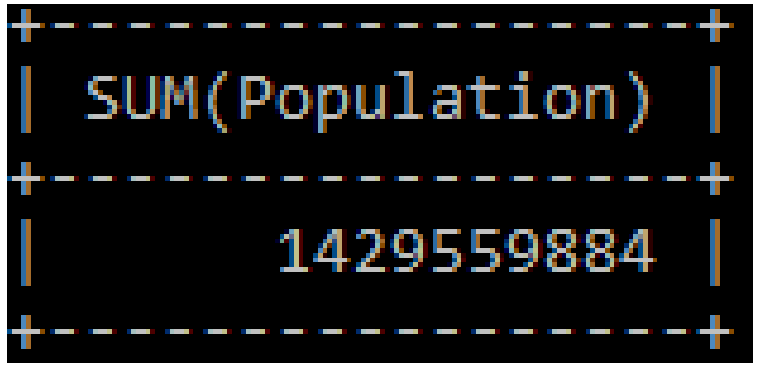
- The SUM() function returns the total sum of a numeric column.

```
SELECT SUM(column_name)
FROM table_name;
```

SUM

- The following SQL statement finds the total **population** from city table:

```
SELECT SUM(Population)
FROM City;
```



```
+-------------------+
| SUM(Population) |
+-------------------+
|      1429559884 |
+-------------------+
```

Purwadhika
Startup and Coding School

SUM

- The following SQL statement finds the total **GNP** from country table:

```
SELECT SUM(GNP)
FROM Country;
```

```
+-------------+
| SUM(GNP)    |
+-------------+
| 29354907.90 |
+-------------+
```

Purwadhika
Startup and Coding School

Reference

https://www.w3schools.com/sql/sql_top.asp

https://www.w3schools.com/sql/sql_count_avg_sum.asp

Purwadhika
Startup and Coding School

SESSIONS 1

INSERT Statement

Data Science Program

Purwadhika
Startup and Coding School

INSERT INTO

- The INSERT INTO statement is used to insert new records in a table.
- It is possible to write the INSERT INTO statement in two ways.

1) First Method to INSERT INTO

- The first way specifies both the column names and the values to be inserted:

```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
```

INSERT INTO

1) First Method to INSERT INTO

INSERT INTO persons (PersonID, LastName, FirstName, Address, City)
VALUES (01, 'Andrew', 'Michael', 'Jln. Mawar', 'BSD');

PersonID	LastName	FirstName	Address	City
1	Andrew	Michael	Jln. Mawar	BSD

Purwadhika
Startup and Coding School

INSERT INTO

2) Second Method to INSERT INTO

- If you are adding values for all the columns of the table, you do not need to specify the column names in the SQL query.
- However, make sure the order of the values is in the same order as the columns in the table.
- The INSERT INTO syntax would be as follows:

```
INSERT INTO table_name
VALUES (value1, value2, value3, ...);
```

INSERT INTO

2) Second Method to INSERT INTO

INSERT INTO persons

VALUES (02, ‘Zidane’, ‘Zinedine’, ‘Jln. Anggrek’, ‘DKI’);

PersonID	LastName	FirstName	Address	City
1	Andrew	Michael	Jln. Mawar	BSD
2	Zidane	Zinedine	Jln. Anggrek	DKI

Purwadhika
Startup and Coding School

Reference

https://www.w3schools.com/sql/sql_insert.asp

Purwadhika
Startup and Coding School

SESSIONS 1

WHERE Statements

Data Science Program

Purwadhika
Startup and Coding School

WHERE

- The WHERE clause is used to filter records.
- The WHERE clause is used to extract only those records that fulfill a specified condition.

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

WHERE

Select all column from City table where Population greater than 1 million.

```
SELECT * FROM City
WHERE Population > 1000000;
```

ID	Name	CountryCode	District	Population
1	Kabul	AFG	Kabol	1780000
35	Alger	DZA	Alger	2168000
56	Luanda	AGO	Luanda	2022000
69	Buenos Aires	ARG	Distrito Federal	2982146
70	La Matanza	ARG	Buenos Aires	1266461

WHERE

ID	Name	CountryCode	District	Population
1901	XiÂ´an	CHN	Shaanxi	2761400
1930	Xuzhou	CHN	Jiangsu	810000
1936	Xining	CHN	Qinghai	700200
1957	Xinxiang	CHN	Henan	473762
1962	Xiangtan	CHN	Hunan	441968
1970	Xiangfan	CHN	Hubei	410407
1990	Xianyang	CHN	Shaanxi	352125
2008	Xingtai	CHN	Hebei	302789
2015	Xintai	CHN	Shandong	281248
2046	Xiantao	CHN	Hubei	222884
2060	Xuchang	CHN	Henan	208815
2076	Xinyang	CHN	Henan	192509
2108	Xinyu	CHN	Jiangxi	173524
2112	Xiaogan	CHN	Hubei	166280
2118	Xiaoshan	CHN	Zhejiang	162930
2120	Xinghua	CHN	Jiangsu	161910
2153	Xianning	CHN	Hubei	136811
2158	Xichang	CHN	Sichuan	134419
2204	Xuangzhou	CHN	Anhui	112673
2227	Xingcheng	CHN	Liaoning	102384
2233	Xinzhou	CHN	Shanxi	98667
2247	Xilin Hot	CHN	Inner Mongolia	90646
2555	Xalapa	MEX	Veracruz	390058
2707	Xai-Xai	MOZ	Gaza	99442

Select all column from City table where City Name is started with ‘X’ character.

SELECT * FROM City

WHERE Name like ‘X%’;

Purwadhika
Startup and Coding School

Reference

https://www.w3schools.com/sql/sql_where.asp

Purwadhika
Startup and Coding School

SESSIONS 1

UPDATE and DELETE Statements

Data Science Program

Purwadhika
Startup and Coding School

UPDATE

Purwadhika
Startup and Coding School

UPDATE

- The UPDATE statement is used to modify the existing records in a table.

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

UPDATE

Before:

PersonID	LastName	FirstName	Address	City
1	Andrew	Michael	Jln. Mawar	BSD
2	Zidane	Zinedine	Jln. Anggrek	DKI

```
UPDATE persons
SET Address = 'Jln. Melati', City= 'DKI'
WHERE PersonID = 1;
```

After:

PersonID	LastName	FirstName	Address	City
1	Andrew	Michael	Jln. Melati	DKI
2	Zidane	Zinedine	Jln. Anggrek	DKI

Purwadhika
Startup and Coding School

UPDATE

Before:

PersonID	LastName	FirstName	Address	City
1	Andrew	Michael	Jln. Melati	DKI
2	Zidane	Zinedine	Jln. Anggrek	DKI

```
UPDATE persons
SET LastName = 'Andrea', FirstName = 'Robert'
WHERE PersonID = 2;
```

After:

PersonID	LastName	FirstName	Address	City
1	Andrew	Michael	Jln. Melati	DKI
2	Andrea	Robert	Jln. Anggrek	DKI

Purwadhika
Startup and Coding School

DELETE

Purwadhika
Startup and Coding School

DELETE

- The DELETE statement is used to delete existing records in a table.

DELETE FROM *table_name*

WHERE *condition*;

DELETE

'Pelanggan' Table

Before:

No	Nama	Usia	Berat	Kota	Tahun	Data_Diinput
2	Andi Santosa	42	78.2	Jakarta	2001	NULL
3	Budi Permana	32	88.1	Jakarta	1999	NULL
4	Cecep Sutisna	35	68.9	Bandung	2000	NULL
5	Dedi Hartanto	32	71.3	Salatiga	1998	NULL
6	Eva Soraya	32	48.1	Medan	2015	NULL
7	Farah Naimah	29	52.2	Surabaya	2010	NULL
8	Gianti Safitri	29	51.6	Bandung	2017	NULL
9	Hamzah Syah	34	66.0	Yogyakarta	2008	NULL
10	Irene Sukindar	25	49.3	Jakarta	2016	NULL
11	Joni Saputra	28	69.8	Yogyakarta	2000	NULL

DELETE

DELETE FROM pelanggan WHERE Usia = 28;

After:

No	Nama	Usia	Berat	Kota	Tahun	Data_Diinput
2	Andi Santosa	42	78.2	Jakarta	2001	NULL
3	Budi Permana	32	88.1	Jakarta	1999	NULL
4	Cecep Sutisna	35	68.9	Bandung	2000	NULL
5	Dedi Hartanto	32	71.3	Salatiga	1998	NULL
6	Eva Soraya	32	48.1	Medan	2015	NULL
7	Farah Naimah	29	52.2	Surabaya	2010	NULL
8	Gianti Safitri	29	51.6	Bandung	2017	NULL
9	Hamzah Syah	34	66.0	Yogyakarta	2008	NULL
10	Irene Sukindar	25	49.3	Jakarta	2016	NULL

Reference

https://www.w3schools.com/sql/sql_update.asp

https://www.w3schools.com/sql/sql_delete.asp

Purwadhika
Startup and Coding School

SESSIONS 1

Using String Patterns and Ranges

Data Science Program

Purwadhika
Startup and Coding School

Using String Patterns

String Patterns using LIKE Operator

- The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.
- There are two wildcards often used in conjunction with the LIKE operator:

  % : The percent sign represents zero, one, or multiple characters

  _ : The underscore represents a single character

The percent sign and the underscore can also be used in combinations!

```
SELECT column1, column2, ...
FROM table_name
WHERE columnN LIKE pattern;
```

Purwadhika
Startup and Coding School

LIKE Operator

LIKE Operator	Description
WHERE SellerName LIKE 'a%'	Finds any values that start with "a"
WHERE SellerName LIKE '%a'	Finds any values that end with "a"
WHERE SellerName LIKE '%or%'	Finds any values that have "or" in any position
WHERE SellerName LIKE '_r%'	Finds any values that have "r" in the second position
WHERE SellerName LIKE 'a_%'	Finds any values that start with "a" and are at least 2 characters in length
WHERE SellerName LIKE 'a__%'	Finds any values that start with "a" and are at least 3 characters in length
WHERE SellerName LIKE 'a%o'	Finds any values that start with "a" and ends with "o"

LIKE Operator

- Show all columns from City Table where District name is started ‘Y’

```
SELECT * FROM CITY
WHERE DISTRICT LIKE ‘Y%’;
```

ID	Name	CountryCode	District	Population
126	Yerevan	ARM	Yerevan	1248700
955	Yogyakarta	IDN	Yogyakarta	418944
1001	Depok	IDN	Yogyakarta	106800
1396	Yazd	IRN	Yazd	326776
1614	Shimonoseki	JPN	Yamaguchi	257263

Purwadhika
Startup and Coding School

LIKE Operator

- Show all columns from City Table where District name is ended with ‘x’

```
SELECT * FROM CITY
WHERE DISTRICT LIKE ‘%x’;
```

ID	Name	CountryCode	District	Population
3350	Sfax	TUN	Sfax	257800

Purwadhika
Startup and Coding School

LIKE Operator

- Show all columns from City Table where City name is started with ‘Y’ and ended with ‘a’

```
SELECT * FROM CITY
WHERE NAME LIKE ‘Y%a’;
```

ID	Name	CountryCode	District	Population
955	Yogyakarta	IDN	Yogyakarta	418944
1568	Yokosuka	JPN	Kanagawa	430200
1615	Yamagata	JPN	Yamagata	255617
1765	Yonezawa	JPN	Yamagata	95592
1767	Yamatokoriyama	JPN	Nara	95165

Purwadhika
Startup and Coding School

Ranges

Purwadhika
Startup and Coding School

BETWEEN

- The BETWEEN operator selects values within a given range. The values can be numbers, text, or dates.
- The BETWEEN operator is inclusive: begin and end values are included.

```
SELECT column_name(s)
FROM table_name
WHERE column_name BETWEEN value1 AND value2;
```

Purwadhika
Startup and Coding School

BETWEEN

- Show **city name and population** from City table where population between 1 – 2 million.

Name	Population
Kabul	1780000
La Matanza	1266461
CÃ³rdoba	1157507
Yerevan	1248700
Brisbane	1291117

```
SELECT Name, Population
FROM City
WHERE Population BETWEEN
1000000 AND 2000000;
```

Purwadhika
Startup and Coding School

BETWEEN

- Show **country name, region, and life expectancy** from Country table where life expectancy between 80-90 years.

Name	Region	LifeExpectancy
Andorra	Southern Europe	83.5
Japan	Eastern Asia	80.7
Macao	Eastern Asia	81.6
Singapore	Southeast Asia	80.1
San Marino	Southern Europe	81.1

```
SELECT Name, Region, LifeExpectancy

FROM Country

WHERE LifeExpectancy BETWEEN 80 AND 90;
```

Purwadhika
Startup and Coding School

NOT BETWEEN

- Show **country name, region, and life expectancy** from Country table where life expectancy not between 45-90 years.

Name	Region	LifeExpectancy
Angola	Central Africa	38.3
Botswana	Southern Africa	39.3
Central African Republic	Central Africa	44.0
Mozambique	Eastern Africa	37.5
Malawi	Eastern Africa	37.6
Namibia	Southern Africa	42.5
Niger	Western Africa	41.3
Rwanda	Eastern Africa	39.3
Swaziland	Southern Africa	40.4
Uganda	Eastern Africa	42.9
Zambia	Eastern Africa	37.2
Zimbabwe	Eastern Africa	37.8

```
SELECT Name, Region, LifeExpectancy

FROM Country

WHERE LifeExpectancy NOT BETWEEN 45 AND 90;
```

Purwadhika
Startup and Coding School

Reference

- https://www.w3schools.com/sql/sql_like.asp
- https://www.w3schools.com/sql/sql_between.asp

SESSIONS 1

Sorting Result Sets

Data Science Program

Purwadhika
Startup and Coding School

ORDER BY

- The ORDER BY keyword is used to sort the result-set in ascending or descending order.
- The ORDER BY keyword sorts the records in ascending order by default. To sort the records in descending order, use the DESC keyword.

```
SELECT column1, column2, ...
FROM table_name
ORDER BY column1, column2, ... ASC/DESC;
```

Purwadhika
Startup and Coding School

ORDER BY

- Show all columns from City Tables and data will be ordered by city name column (ascending).

```
SELECT * FROM CITY
ORDER BY Name;
```

ID	Name	CountryCode	District	Population
670	A CoruÃ±a (La CoruÃ±a)	ESP	Galicia	243402
3097	Aachen	DEU	Nordrhein-Westfalen	243825
3318	Aalborg	DNK	Nordjylland	161161
2760	Aba	NGA	Imo & Abia	298900
1404	Abadan	IRN	Khuzestan	206073

Purwadhika
Startup and Coding School

ORDER BY

- Show all columns from City Tables and data will be ordered by city name column (descending).

```
SELECT * FROM CITY
ORDER BY Name DESC;
```

ID	Name	CountryCode	District	Population
698	[San CristÃ³bal de] la Laguna	ESP	Canary Islands	127945
3446	Zytomyr	UKR	Zytomyr	297000
28	Zwolle	NLD	Overijssel	105819
3145	Zwickau	DEU	Saksi	104146
2025	Zunyi	CHN	Guizhou	261862

Purwadhika
Startup and Coding School

Reference

- https://www.w3schools.com/sql/sql_orderby.asp

SESSIONS 1

Grouping Result Sets

Data Science Program

Purwadhika
Startup and Coding School

GROUP BY

- The GROUP BY statement groups rows that have the same values into summary rows, like "find the number of customers in each country".
- The GROUP BY statement is often used with aggregate functions (COUNT, MAX, MIN, SUM, AVG) to group the result-set by one or more columns.

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
ORDER BY column_name(s);
```

Purwadhika
Startup and Coding School

GROUP BY

- Show how many city name in every country code based on City ID

```
SELECT COUNT(ID), CountryCode
FROM City
GROUP BY CountryCode;
```

COUNT(ID)	CountryCode
1	ABW
4	AFG
5	AGO
2	AIA
1	ALB

Purwadhika
Startup and Coding School

GROUP BY & WHERE

- Show average city population in every district in Indonesia (IDN)

```
SELECT AVG(Population), District
FROM CITY
WHERE CountryCode = 'IDN'
GROUP BY District;
```

AVG(Population)	District
9604900.0000	Jakarta Raya
377868.3846	East Java
321531.4400	West Java
373713.8571	Sumatera Utara
673382.0000	Sumatera Selatan
286595.7273	Central Java
1060257.0000	Sulawesi Selatan
680332.0000	Lampung
534474.0000	Sumatera Barat
482931.0000	Kalimantan Selatan
206803.0000	Riau
435000.0000	Bali
262872.0000	Yogyakarta
409632.0000	Kalimantan Barat
368963.5000	Kalimantan Timur
385201.0000	Jambi
213173.0000	Sulawesi Utara
306600.0000	Nusa Tenggara Barat
249312.0000	Molukit
146439.0000	Bengkulu
126504.5000	Aceh
142800.0000	Sulawesi Tengah
129300.0000	Nusa Tenggara Timur
99693.0000	Kalimantan Tengah
94800.0000	Sulawesi Tenggara
94700.0000	West Irian

GROUP BY, WHERE, & AS

- Show average city population in every district in Indonesia (IDN).
- AS is Alias syntax to give new column name in result set.

```
SELECT AVG(Population) AS Rata_rata,
District AS Provinsi
FROM CITY
WHERE CountryCode = 'IDN'
GROUP BY District;
```

Rata_rata	Provinsi
9604900.0000	Jakarta Raya
377868.3846	East Java
321531.4400	West Java
373713.8571	Sumatera Utara
673382.0000	Sumatera Selatan
286595.7273	Central Java
1060257.0000	Sulawesi Selatan
680332.0000	Lampung
534474.0000	Sumatera Barat
482931.0000	Kalimantan Selatan
206803.0000	Riau
435000.0000	Bali
262872.0000	Yogyakarta
409632.0000	Kalimantan Barat
368963.5000	Kalimantan Timur
385201.0000	Jambi
213173.0000	Sulawesi Utara
306600.0000	Nusa Tenggara Barat
249312.0000	Molukit
146439.0000	Bengkulu
126504.5000	Aceh
142800.0000	Sulawesi Tengah
129300.0000	Nusa Tenggara Timur
99693.0000	Kalimantan Tengah
94800.0000	Sulawesi Tenggara
94700.0000	West Irian

GROUP BY, WHERE, AS, & HAVING

- Show average city population in every district in Indonesia (IDN), but only showing the average greater than 500.000.
- **HAVING** is similar with WHERE which is syntax to filter result set. But, HAVING is used after grouping (GROUP BY).

```
SELECT AVG(Population) AS Rata_rata,
District AS Provinsi
FROM CITY
WHERE CountryCode = 'IDN'
GROUP BY District
HAVING Rata_rata > 500000;
```

Rata_rata	Provinsi
9604900.0000	Jakarta Raya
673382.0000	Sumatera Selatan
1060257.0000	Sulawesi Selatan
680332.0000	Lampung
534474.0000	Sumatera Barat

Purwadhika
Startup and Coding School

GROUP BY, WHERE, AS, & HAVING

- Show average city population in every district in Indonesia (IDN), but only showing the ‘Provinsi’ which is started with ‘K’.

```
SELECT AVG(Population) AS Rata_rata,
District AS Provinsi
FROM CITY
WHERE CountryCode = ‘IDN’
GROUP BY District
HAVING Provinsi LIKE ‘K%;
```

Rata_rata	Provinsi
482931.0000	Kalimantan Selatan
409632.0000	Kalimantan Barat
368963.5000	Kalimantan Timur
99693.0000	Kalimantan Tengah

Purwadhika
Startup and Coding School

Reference

- https://www.w3schools.com/sql/sql_groupby.asp

SESSIONS 1

Built-in Database Functions

Data Science Program

Purwadhika
Startup and Coding School

Built-in Database Functions

Most databases come with built-in function. This functions can be included in SQL statements. It will be significantly reduce the amount of data that needs to be retrieved, and also speed up data processing.

- **Aggregate Functions**: This function will perform on collection of values or entire column. The output is single value. Example functions: SUM, MIN, MAX, AVG.
- **Scalar Function**: This function will perform operations on every individual value. Example functions: ROUND, LENGTH, UCASE, LCASE.

AGGREGATE FUNCTIONS

Purwadhika
Startup and Coding School

SUM

The SUM() function returns the total sum of a numeric column.

```
SELECT SUM(column_name)
FROM table_name
WHERE condition;
```

Example:

```
SELECT SUM(Population) AS
Total_Population
FROM City
WHERE CountryCode = ‘IND’;
```

```
mysql> SELECT SUM(Population) as Total_Population
    -> FROM City
    -> WHERE CountryCode = 'IND';
+------------------+
| Total_Population |
+------------------+
|        123298526 |
+------------------+
```

Purwadhika
Startup and Coding School

COUNT

The COUNT() function returns the number of rows that matches a specified criterion.

```
SELECT COUNT(column_name)
FROM table_name
WHERE condition;
```

Example:

```
SELECT COUNT(name) AS
Total_City
FROM City
WHERE CountryCode = ‘IDN’;
```

```
mysql> SELECT COUNT(Name) AS Total_City
    -> FROM CITY
    -> WHERE CountryCode = 'IDN';
+------------+
| Total_City |
+------------+
|         85 |
+------------+
```

Purwadhika
Startup and Coding School

AVG

The AVG() function returns the average value of a numeric column.

```
SELECT AVG(column_name)
FROM table_name
WHERE condition;
```

Example:

```
SELECT AVG(Population) AS
Avg_Population
FROM City
WHERE CountryCode = ‘IDN’;
```

```
mysql> SELECT AVG(Population) AS Avg_Population
    -> FROM City
    -> WHERE CountryCode = 'IDN';
+----------------+
| Avg_Population |
+----------------+
|    441008.1765 |
+----------------+
```

Purwadhika
Startup and Coding School

MIN

The MIN() function returns the smallest value of the selected column.

```
SELECT MIN(column_name)
FROM table_name
WHERE condition;
```

Example:

```
SELECT MIN(Population) AS
Min_Population
FROM City
WHERE CountryCode = ‘IDN’;
```

```
mysql> SELECT MIN(Population) AS Min_Population
    -> FROM City
    -> WHERE CountryCode = 'IDN';
+----------------+
| Min_Population |
+----------------+
|          89900 |
+----------------+
```

Purwadhika
Startup and Coding School

MAX

The MAX() function returns the largest value of the selected column.

```
SELECT MAX(column_name)
FROM table_name
WHERE condition;
```

Example:

```
SELECT MAX(Population) AS
Max_Population
FROM City
WHERE CountryCode = ‘IDN’;
```

```
mysql> SELECT MAX(Population) AS Max_Population
    -> FROM City
    -> WHERE CountryCode = 'IDN';
+----------------+
| Max_Population |
+----------------+
|        9604900 |
+----------------+
```

Purwadhika
Startup and Coding School

SCALAR FUNCTIONS

Purwadhika
Startup and Coding School

ROUND

The ROUND() function rounds a number to a specified number of decimal places.

ROUND(*number*, *decimals*)

- **Number:** Required. The number to be rounded
- **Decimal:** Optional. The number of decimal places to round *number* to. If omitted, it returns the integer (no decimals)

NAME	ROUND(LifeExpectancy)
Aruba	78
Afghanistan	46
Angola	38
Anguilla	76
Albania	72

```
SELECT NAME,
ROUND(LifeExpectancy)
FROM Country;
```

Purwadhika
Startup and Coding School

ROUND

- Show population density (Population/Surface Area) only in Southeast Asia Region. The value of population density is rounded 2 decimal after coma.

NAME	REGION	Population_Density
Brunei	Southeast Asia	56.90
Indonesia	Southeast Asia	111.37
Cambodia	Southeast Asia	61.69
Laos	Southeast Asia	22.94
Myanmar	Southeast Asia	67.41
Malaysia	Southeast Asia	67.46
Philippines	Southeast Asia	253.22
Singapore	Southeast Asia	5771.84
Thailand	Southeast Asia	119.66
East Timor	Southeast Asia	59.50
Vietnam	Southeast Asia	240.68

```
SELECT NAME, REGION,
ROUND(Population/SurfaceArea, 2)
AS Population_Density
FROM Country
WHERE Region = 'Southeast Asia';
```

LENGTH

The LENGTH() function returns the length of a string (in bytes).

LENGTH(string)

- **String:** Required. The string to count the length for

Name	Length_Name
Philippines	11
East Timor	10
Indonesia	9
Singapore	9
Cambodia	8
Malaysia	8
Thailand	8
Myanmar	7
Vietnam	7
Brunei	6
Laos	4

```
SELECT Name, LENGTH(Name) AS Length_Name
FROM Country
WHERE Region = ‘ Southeast Asia’
ORDER BY Length_Name DESC;
```

Purwadhika
Startup and Coding School

UCASE

The UCASE() function converts a string to upper-case. This function is equal to the UPPER() function.

UCASE(text)

- **String:** Required. The string to convert

UCASE(Name)	Population
INDONESIA	212107000
VIETNAM	79832000
PHILIPPINES	75967000
THAILAND	61399000
MYANMAR	45611000
MALAYSIA	22244000
CAMBODIA	11168000
LAOS	5433000
SINGAPORE	3567000
EAST TIMOR	885000
BRUNEI	328000

```
SELECT UCASE(Name), Population
FROM Country
WHERE Region = 'Southeast Asia'
ORDER BY Population DESC;
```

Purwadhika
Startup and Coding School

LCASE

The LCASE() function converts a string to lower-case. The LOWER() function is a synonym for the LCASE() function.

LCASE(text)

- **String:** Required. The string to convert

LCASE(Name)	Population
indonesia	212107000
vietnam	79832000
philippines	75967000
thailand	61399000
myanmar	45611000
malaysia	22244000
cambodia	11168000
laos	5433000
singapore	3567000
east timor	885000
brunei	328000

```
SELECT LCASE(Name), Population
FROM Country
WHERE Region = 'Southeast Asia'
ORDER BY Population DESC;
```

Purwadhika
Startup and Coding School

Reference

- https://www.w3schools.com/sql/sql_count_avg_sum.asp
- https://www.w3schools.com/sql/sql_min_max.asp
- https://www.w3schools.com/sql/func_mysql_round.asp
- https://www.w3schools.com/sql/func_mysql_ucase.asp
- https://www.w3schools.com/sql/func_mysql_lcase.asp

SESSIONS 1

Date and Time Built-in Functions

Data Science Program

Purwadhika
Startup and Coding School

Date & Time Functions

Most databases contain special data types for date and times. This is format for date and time:

DATE: YYYYMMDD

TIME: HHMMSS

TIMESTAMP: YYYYXXDDHHMMSSZZZZZZ

Date and Time Functions

YEAR(), MONTH(), DAY(), DAYOFMONTH(), DAYOFWEEK(), DAYOFYEAR(), WEEK(), HOUR(), MINUTE(), SECOND()

DAY

- Show **customer_id and day of payment date** from Payment table where amount above US$ 11.

```
SELECT Customer_id, DAY(payment_date)
FROM PAYMENT
WHERE amount > 11;
```

Customer_id	DAY(payment_date)
13	29
116	21
195	23
196	25
204	22
237	2
305	17
362	21
591	7
592	6

DAYNAME

- Show **average amount based on day** from Payment table and ordered by average amount.

```
SELECT AVG(amount) AS Average_Amount,
DAYNAME(payment_date) AS Day
FROM PAYMENT
GROUP BY DAYNAME(payment_date)
ORDER BY Average_Amount;
```

Average_Amount	Day
4.146187	Tuesday
4.153156	Wednesday
4.159114	Monday
4.200776	Sunday
4.208654	Friday
4.243636	Thursday
4.296096	Saturday

MONTH

- Show **total amount** from Payment table only on August.

```
SELECT SUM(amount) AS Total_Amount_August
FROM PAYMENT
WHERE MONTH(payment_date) = 8;
```

Total_Amount_August
24072.13

Purwadhika
Startup and Coding School

MONTHNAME

- Show **average amount based on month** from Payment table and ordered by average amount.

```
SELECT AVG(amount) AS Average_Amount,
MONTHNAME(payment_date) AS Month_Name
FROM PAYMENT
GROUP BY Month_Name
ORDER BY Average_Amount;
```

Average_Amount	Month_Name
2.825165	February
4.166038	June
4.169775	May
4.227968	July
4.232835	August

Purwadhika
Startup and Coding School

DATE ARITHMETIC

- Show **day name after one day payment** from Payment table where staff_id is 1 and amount greater than US$ 11

```
SELECT Customer_id, Amount,
DAYNAME(payment_date + 1) AS One_Day_After_Payment
FROM PAYMENT
WHERE staff_id = 1 AND
amount > 11;
```

Customer_id	Amount	One_Day_After_Payment
305	11.99	Friday
362	11.99	Sunday
592	11.99	Wednesday

Purwadhika
Startup and Coding School

YEAR

- Show **average amount every year** from Payment table.

```
SELECT YEAR(payment_date) AS Year_Sales,
AVG(amount) AS Total_Amount_Yearly
FROM PAYMENT
GROUP BY Year_sales;
```

Year_Sales	Average_Amount_Yearly
2005	4.216445
2006	2.825165

Purwadhika
Startup and Coding School

Reference

- https://www.w3schools.com/sql/func_mysql_day.asp
- https://www.w3schools.com/sql/func_mysql_dayname.asp
- https://www.w3schools.com/sql/func_mysql_month.asp
- https://www.w3schools.com/sql/func_sqlserver_year.asp

SESSIONS 1

Sub-Queries and Nested Select

Data Science Program

Purwadhika
Startup and Coding School

Sub-Queries

Sub-Query: A query inside another SQL query.

- A sub-query is a SQL query nested inside a larger query.
- The sub-query can be nested inside a SELECT, INSERT, UPDATE, or DELETE statement or inside another sub-query.
- Sub-query is usually added within the WHERE Clause of another SQL SELECT statement.

Sub-Queries

EXAMPLE SYNTAX

```
select COLUMN1 from TABLE
    where COLUMN 2 = (select MAX(COLUMN2) from TABLE)
```

```
select ID, NAME, SALARY from EMPLOYEE
    where SALARY < (select AVG(SALARY) from employees)
```

Purwadhika
Startup and Coding School

Sub-Queries & Aggregate Functions

We cannot evaluate Aggregate function like AVG() in the WHERE clause. Therefore, use a sub-Select expression like example bellow:

```
select ID, NAME, SALARY
    from EMPLOYEE
    where SALARY <
    (select AVG(SALARY) from employees)
```

Sub-Queries & Aggregate Functions

Show **employees name where their age is above average age of all employees age** from new_employees table.

```
select First_Name, Last_Name, Age
from new_employees
where Age >
(select avg(Age) from new_employees);
```

first_name	last_name	age
Alain	Chappelet	47
Alejandro	McAlpine	47
Amabile	Gomatam	45
Anneke	Preusig	47
Arif	Merlo	44
Arumugam	Ossenbru…	43
Berhard	McFarlin	46

Purwadhika
Startup and Coding School

Sub-Queries in list of columns

We can substitute column name with a sub-query. It's called column expressions. For example, see this syntax:

```
select ID, SALARY,
    (select AVG (SALARY) from employees)
            AS Average_Salary
    from employees;
```

Sub-Queries in list of columns

Show **employees name, their age, and also the youngest age and the oldest age** from new_employees table.

```
select First_Name, Last_Name, Age,
(select min(Age) from new_employees) Youngest,
(select max(Age) from new_employees) Oldest
from new_employees;
```

first_name	last_name	age	Youngest	Oldest
Georgi	Facello	47	35	48
Bezalel	Simmel	36	35	48
Parto	Bamford	41	35	48
Chirstian	Koblick	46	35	48
Kyoichi	Maliniak	45	35	48
Anneke	Preusig	47	35	48
Tzvetan	Zielinski	43	35	48
Saniya	Kalloufi	42	35	48
Sumant	Peac	48	35	48
Duangkaew	Piveteau	37	35	48

Sub-Queries in FROM Clause

We also can substitute the TABLE name with a sub-query. It's called Derived Tables or Table Expressions.

```
select * from
      ( select ID, NAME, DEPARTMENT_ID
            from employees) AS ALL_EMPLOYEES
```

Sub-Queries in FROM Clause

Substitute the **table name from 'employees' to be 'employee_biodata'** with a sub-query

```
select *from
    (select First_name, Last_name, Gender, Birth_date
    from employees)
        as Employee_Biodata;
```

First_name	Last_name	Gender	Birth_date
Georgi	Facello	M	1953-09-02
Bezalel	Simmel	F	1964-06-02
Parto	Bamford	M	1959-12-03
Chirstian	Koblick	M	1954-05-01
Kyoichi	Maliniak	M	1955-01-21
Anneke	Preusig	F	1953-04-20
Tzvetan	Zielinski	F	1957-05-23
Saniya	Kalloufi	M	1958-02-19
Sumant	Peac	F	1952-04-19
Duangkaew	Piveteau	F	1963-06-01

Reference

- https://www.w3resource.com/sql/subqueries/understanding-sql-subqueries.php

Purwadhika
Startup and Coding School

SESSIONS 1

Working with Multiple Tables

Data Science Program

Purwadhika
Startup and Coding School

Access Multiple Tables

Methods to access multiple tables in the same query

1. Sub-queries
2. Implicit JOIN
3. JOIN operators (INNER JOIN, OUTER JOIN, etc.)

Sub-Queries

Access Multiple Tables

Purwadhika
Startup and Coding School

Sub-Queries

Sub-queries can be used to access multiple tables. For example, we can access EMPLOYEE & TITLE table.

emp_no	birth_date	first_name	last_name	gender	hire_date
10001	1953-09-02	Georgi	Facello	M	1986-06-26
10002	1964-06-02	Bezalel	Simmel	F	1985-11-21
10003	1959-12-03	Parto	Bamford	M	1986-08-28
10004	1954-05-01	Chirstian	Koblick	M	1986-12-01
10005	1955-01-21	Kyoichi	Maliniak	M	1989-09-12
10006	1953-04-20	Anneke	Preusig	F	1989-06-02
10007	1957-05-23	Tzvetan	Zielinski	F	1989-02-10
10008	1958-02-19	Saniya	Kalloufi	M	1994-09-15
10009	1952-04-19	Sumant	Peac	F	1985-02-18
10010	1963-06-01	Duangkaew	Piveteau	F	1989-08-24

Employees Tables

emp_no	title	from_date	to_date
10001	Senior Engineer	1986-06-26	9999-01-01
10002	Staff	1996-08-03	9999-01-01
10003	Senior Engineer	1995-12-03	9999-01-01
10004	Engineer	1986-12-01	1995-12-01
10004	Senior Engineer	1995-12-01	9999-01-01
10005	Senior Staff	1996-09-12	9999-01-01
10005	Staff	1989-09-12	1996-09-12
10006	Senior Engineer	1990-08-05	9999-01-01
10007	Senior Staff	1996-02-11	9999-01-01
10007	Staff	1989-02-10	1996-02-11

Titles Tables

Sub-Queries

Retrieve the **employee** records that correspond to **emp_no** in the TITLES table:

```
select * from employees
    where EMP_NO IN
    (select EMP_NO from titles);
```

emp_no	birth_date	first_name	last_name	gender	hire_date
10001	1953-09-02	Georgi	Facello	M	1986-06-26
10002	1964-06-02	Bezalel	Simmel	F	1985-11-21
10003	1959-12-03	Parto	Bamford	M	1986-08-28
10004	1954-05-01	Chirstian	Koblick	M	1986-12-01
10005	1955-01-21	Kyoichi	Maliniak	M	1989-09-12
10006	1953-04-20	Anneke	Preusig	F	1989-06-02
10007	1957-05-23	Tzvetan	Zielinski	F	1989-02-10
10008	1958-02-19	Saniya	Kalloufi	M	1994-09-15
10009	1952-04-19	Sumant	Peac	F	1985-02-18
10010	1963-06-01	Duangkaew	Piveteau	F	1989-08-24

Sub-Queries

Retrieve the **employee** records that correspond to **emp_no** in the TITLES table and **only has 'Senior Staff'** title:

```
select * from employees
    where EMP_NO IN
    (select EMP_NO from titles
    where title = 'Senior Staff');
```

emp_no	birth_date	first_name	last_name	gender	hire_date
10005	1955-01-21	Kyoichi	Maliniak	M	1989-09-12
10007	1957-05-23	Tzvetan	Zielinski	F	1989-02-10
10013	1963-06-07	Eberhardt	Terkki	M	1985-10-20
10015	1959-08-19	Guoxiang	Nooteboom	M	1987-07-02
10017	1958-07-06	Cristinel	Bouloucos	F	1993-08-03
10036	1959-08-10	Adamantios	Portugali	M	1992-01-03
10038	1960-07-20	Huan	Lortz	M	1989-09-20
10039	1959-10-01	Alejandro	Brender	M	1988-01-19
10041	1959-08-27	Uri	Lenart	F	1989-11-12
10042	1956-02-26	Magy	Stamatiou	F	1993-03-21

Sub-Queries

Retrieve the **employee** records that have **salary above US$ 60K** in SALARIES table:

```
select first_name, last_name, gender, birth_date
from employees
    where EMP_NO IN
    (select EMP_NO from salaries
    where salary > 60000);
```

first_name	last_name	gender	birth_date
Georgi	Facello	M	1953-09-02
Bezalel	Simmel	F	1964-06-02
Chirstian	Koblick	M	1954-05-01
Kyoichi	Maliniak	M	1955-01-21
Anneke	Preusig	F	1953-04-20
Tzvetan	Zielinski	F	1957-05-23
Sumant	Peac	F	1952-04-19
Duangkaew	Piveteau	F	1963-06-01
Eberhardt	Terkki	M	1963-06-07
Berni	Genin	M	1956-02-12

Implicit Join

Access Multiple Tables

Purwadhika
Startup and Coding School

Implicit JOIN

Specify 2 tables in the FROM clause:

```
Select * from employees, salaries;
```

The result is a full join (or Cartesian join):

- Every row in the first table is joined with every row in the second table
- The result set will have more rows than in both tables

Implicit JOIN

Use additional operands to limit the result set:

```
select * from employees, salaries
where employees.emp_no = salaries.emp_no;
```

Use shorter aliases for table names:

```
select * from employees E, salaries S
where E.emp_no = S.emp_no;
```

Implicit JOIN

See **first_name, last_name, and salary** from EMPLOYEE & SALARIES tables.
Column names in the select clause can be pre-fixed by aliases:

```
select E.first_name, E.last_name, S.salary
from employees E, salaries S
where E.emp_no = S.emp_no;
```

first_name	last_name	salary
Florian	Syrotiuk	39507
Divier	Reistad	39520
Basil	Tramer	39735
Pradeep	Makrucki	39765
Shahaf	Famili	39935
Anneke	Preusig	40000
Patricio	Bridgland	40000
Eberhardt	Terkki	40000

Reference

- https://www.w3resource.com/sql/subqueries/understanding-sql-subqueries.php
- https://www.w3schools.com/sql/sql_join.asp

Purwadhika
Startup and Coding School

SESSIONS 1

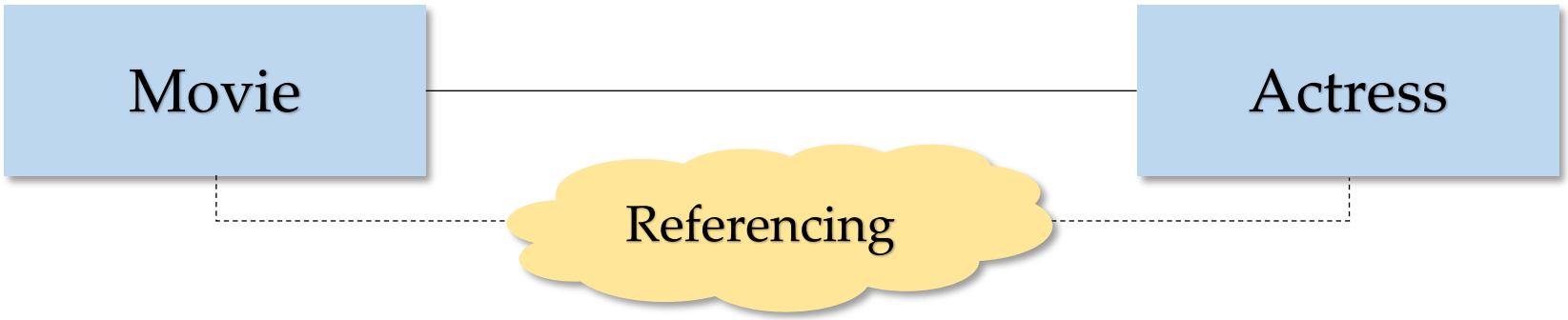
Relational Model Constraints

Data Science Program

Purwadhika
Startup and Coding School

Referencing

- In any business, data must adhere to certain restrictions or rules.
- For example, relationship between actress and movie.
- At least, **one actress** to be actor in **one movie.** This is a one to **one relationship**.
- To look up the actress information, the movie entity refers to the actress entity. To look up the movie information, the actress entity refers to the movie entity. In a relational data model this is called **referencing.**

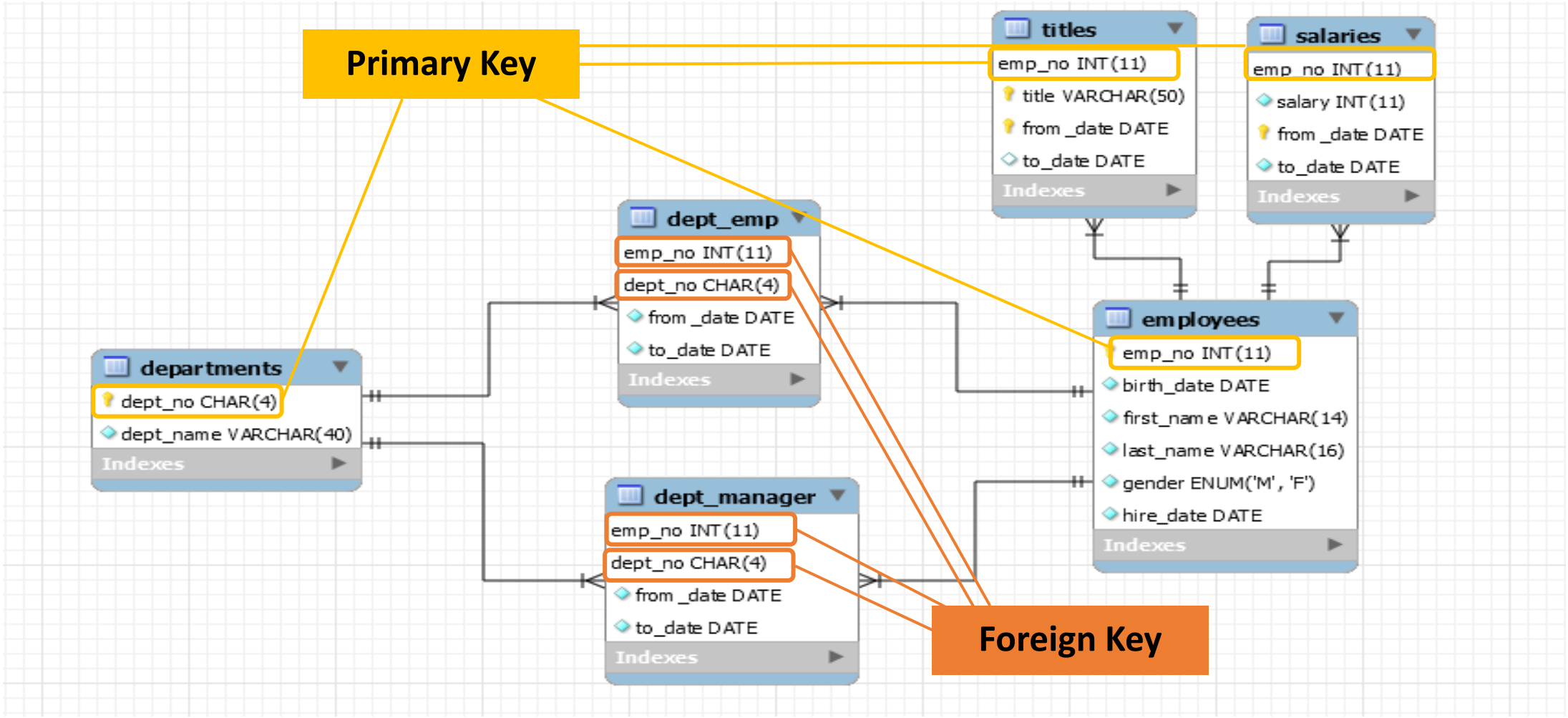


Purwadhika
Startup and Coding School

Primary Key & Foreign Key

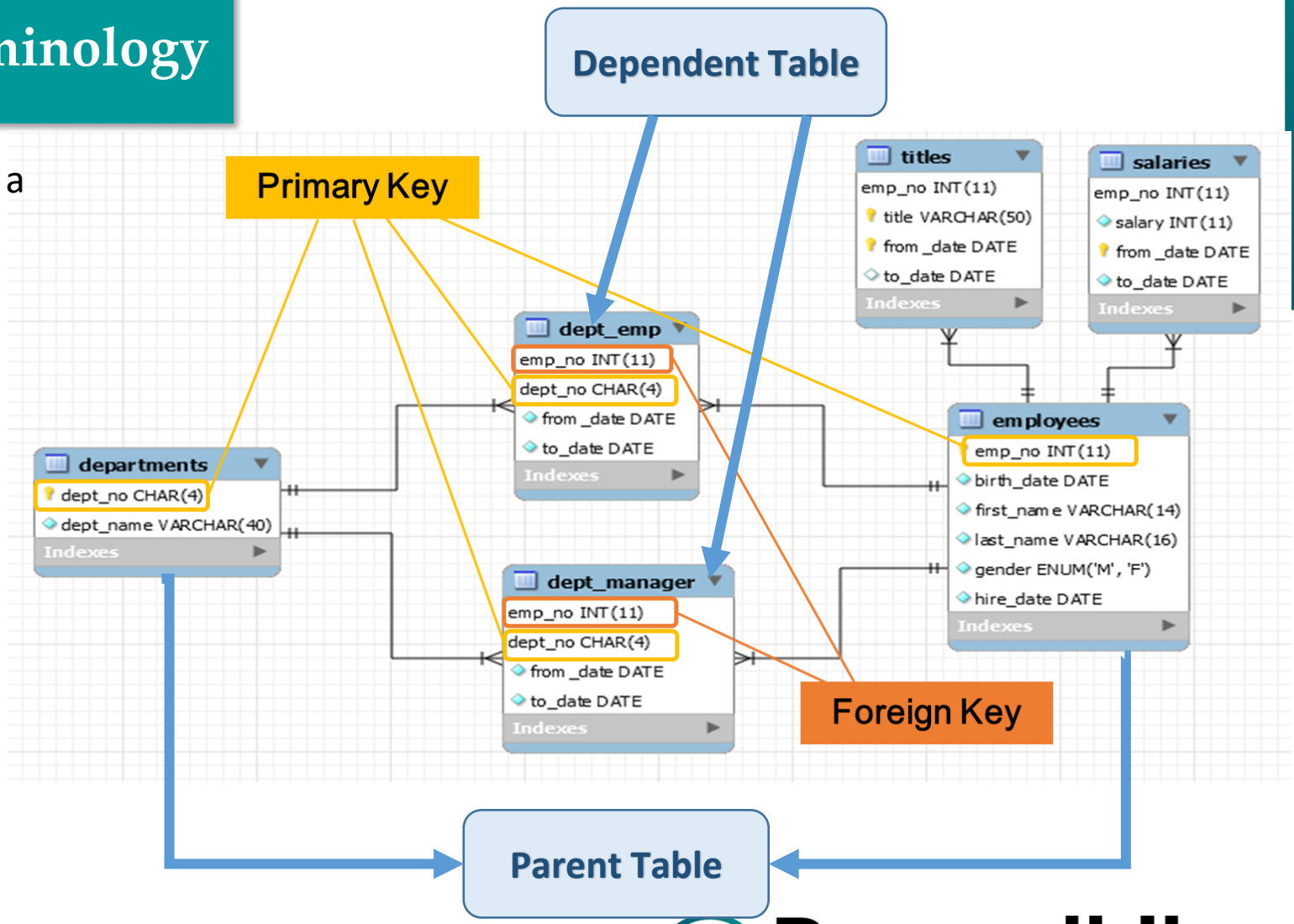
- Every table have **unique identity**, such as salary_id, employee_id, department_id, etc.This identifies the primary key
- The primary key constraint uniquely identifies each record in a table.
- Primary keys must contain UNIQUE values, and cannot contain NULL values.
- But, sometimes **“A” table primary key** is placed in **“B” table**. This ‘primary key’ is called **foreign key** in “B” table.
- A foreign key is a set of columns referring to a primary key of another table. A foreign key is a key **used to link two tables together**.
- A foreign key is a field (or collection of fields) in one table that refers to the primary key in another table.

ERD Representation of a Relational Data Model



Relational Model Terminology

- **Parent table:** a table containing a Primary Key that is related to at least one Foreign Key.
- **Dependent table:** a table containing one or more Foreign Keys



Purwadhika
Startup and Coding School

Reference

- https://www.w3schools.com/sql/sql_primarykey.ASP
- https://www.w3schools.com/sql/sql_foreignkey.asp

Purwadhika
Startup and Coding School

SESSIONS 1

JOIN Table

Data Science Program

Purwadhika
Startup and Coding School

JOIN Table

- A JOIN clause is used to combine rows from two or more tables, based on a related column between them.
- Here are the different types of the JOINs in SQL:
    1) **(INNER) JOIN**: Returns records that have matching values in both tables
    2) **LEFT (OUTER) JOIN**: Returns all records from the left table, and the matched records from the right table
    3) **RIGHT (OUTER) JOIN**: Returns all records from the right table, and the matched records from the left table
    4) **FULL (OUTER) JOIN**: Returns all records when there is a match in either left or right table

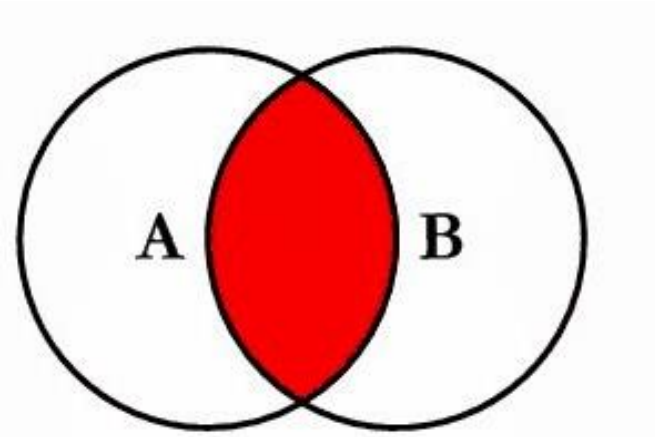
Inner JOIN

Purwadhika
Startup and Coding School

Inner JOIN

- The INNER JOIN keyword selects records that have matching values in both tables.

```
SELECT column_name(s)
FROM table1
INNER JOIN table2
ON table1.column_name = table2.column_name;
```



Purwadhika
Startup and Coding School

Inner JOIN

- **Employees table** INNER JOIN with **Salaries table** based on **emp_no**.

emp_no	birth_date	first_name	last_name	gender	hire_date
10001	1953-09-02	Georgi	Facello	M	1986-06-26
10002	1964-06-02	Bezalel	Simmel	F	1985-11-21
10003	1959-12-03	Parto	Bamford	M	1986-08-28
10004	1954-05-01	Chirstian	Koblick	M	1986-12-01
10005	1955-01-21	Kyoichi	Maliniak	M	1989-09-12
10006	1953-04-20	Anneke	Preusig	F	1989-06-02
10007	1957-05-23	Tzvetan	Zielinski	F	1989-02-10
10008	1958-02-19	Saniya	Kalloufi	M	1994-09-15
10009	1952-04-19	Sumant	Peac	F	1985-02-18

Employees tables

emp_no	salary	from_date	to_date
10001	60117	1986-06-26	1987-06-26
10001	62102	1987-06-26	1988-06-25
10001	66074	1988-06-25	1989-06-25
10001	66596	1989-06-25	1990-06-25
10001	66961	1990-06-25	1991-06-25
10001	71046	1991-06-25	1992-06-24
10001	74333	1992-06-24	1993-06-24
10001	75286	1993-06-24	1994-06-24
10001	75994	1994-06-24	1995-06-24

Salaries tables

Inner JOIN

```
select employees.First_name, employees.Last_name, salaries.salary
from employees JOIN salaries
ON employees.emp_no = salaries.emp_no;
```

First_name	Last_name	salary
Florian	Syrotiuk	39507
Divier	Reistad	39520
Basil	Tramer	39735
Pradeep	Makrucki	39765
Shahaf	Famili	39935
Anneke	Preusig	40000
Patricio	Bridgland	40000
Eberhardt	Terkki	40000
Guoxiang	Nooteboom	40000

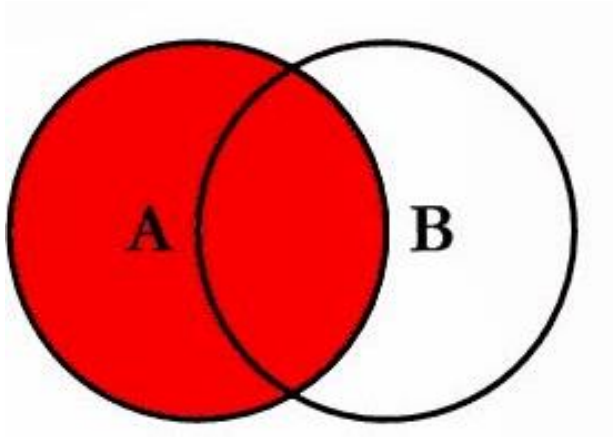
Left JOIN

Purwadhika
Startup and Coding School

Left JOIN

- The LEFT JOIN keyword returns all records from the left table (table1), and the matched records from the right table (table2). The result is NULL from the right side, if there is no match.
- In some databases LEFT JOIN is called LEFT OUTER JOIN.

```
SELECT column_name(s)
FROM table1
LEFT JOIN table2
ON table1.column_name = table2.column_name;
```



Purwadhika
Startup and Coding School

Left JOIN

- **New _employees table** LEFT JOIN with **Dept_info table** based on **emp_no**.

emp_no	first_name	last_name	gender	hire_date	age
10001	Georgi	Facello	M	1986-06-26	47
10002	Bezalel	Simmel	F	1985-11-21	36
10003	Parto	Bamford	M	1986-08-28	41
10004	Chirstian	Koblick	M	1986-12-01	46
10005	Kyoichi	Maliniak	M	1989-09-12	45
10006	Anneke	Preusig	F	1989-06-02	47
10007	Tzvetan	Zielinski	F	1989-02-10	43
10008	Saniya	Kalloufi	M	1994-09-15	42
10009	Sumant	Peac	F	1985-02-18	48

New_employees tables

emp_no	dept_name
10011	Customer Service
10038	Customer Service
10049	Customer Service
10060	Customer Service
10088	Customer Service
10098	Customer Service
10112	Customer Service
10115	Customer Service
10126	Customer Service
10128	Customer Service

Dept_info tables

Purwadhika
Startup and Coding School

Left JOIN

```
select A.first_name, A.last_name, B.dept_name
from new_employees A LEFT JOIN dept_info B
ON A.emp_no = B.emp_no;
```

first_name	last_name	dept_name
Georgi	Facello	Development
Bezalel	Simmel	Sales
Parto	Bamford	Production
Chirstian	Koblick	Production
Kyoichi	Maliniak	Human Resources
Anneke	Preusig	Development
Tzvetan	Zielinski	Research
Saniya	Kalloufi	Development
Sumant	Peac	Quality Management
Duangkaew	Piveteau	Production

Purwadhika
Startup and Coding School

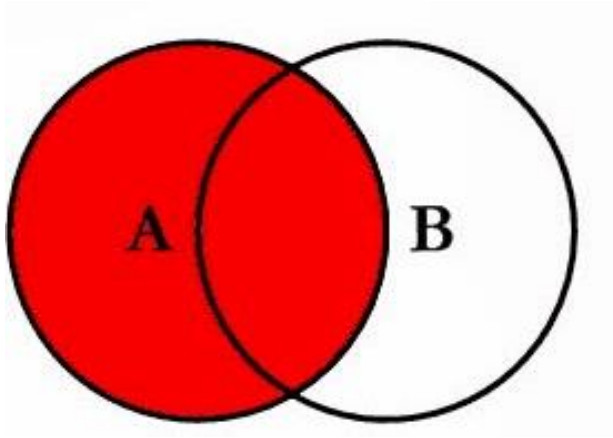
Right JOIN

Purwadhika
Startup and Coding School

Right JOIN

- The RIGHT JOIN keyword returns all records from the right table (table2), and the matched records from the left table (table1). The result is NULL from the left side, when there is no match.
- In some databases RIGHT JOIN is called RIGHT OUTER JOIN.

```
SELECT column_name(s)
FROM table1
RIGHT JOIN table2
ON table1.column_name = table2.column_name;
```



Purwadhika
Startup and Coding School

Right JOIN

- **New _employees table** RIGHT JOIN with **Dept_info table** based on **emp_no**.

emp_no	first_name	last_name	gender	hire_date	age
10001	Georgi	Facello	M	1986-06-26	47
10002	Bezalel	Simmel	F	1985-11-21	36
10003	Parto	Bamford	M	1986-08-28	41
10004	Chirstian	Koblick	M	1986-12-01	46
10005	Kyoichi	Maliniak	M	1989-09-12	45
10006	Anneke	Preusig	F	1989-06-02	47
10007	Tzvetan	Zielinski	F	1989-02-10	43
10008	Saniya	Kalloufi	M	1994-09-15	42
10009	Sumant	Peac	F	1985-02-18	48

New_employees tables

emp_no	dept_name
10011	Customer Service
10038	Customer Service
10049	Customer Service
10060	Customer Service
10088	Customer Service
10098	Customer Service
10112	Customer Service
10115	Customer Service
10126	Customer Service
10128	Customer Service

Dept_info tables

Purwadhika
Startup and Coding School

Right JOIN

```
select A.first_name, A.last_name, B.dept_name
from new_employees A RIGHT JOIN dept_info B
ON A.emp_no = B.emp_no;
```

first_name	last_name	dept_name
Mary	Sluis	Customer Service
Huan	Lortz	Customer Service
Basil	Tramer	Customer Service
Breannda	Billingsley	Customer Service
Jungsoon	Syrzycki	Customer Service
Sreekrishna	Servieres	Customer Service
Yuichiro	Swick	Customer Service
Chikara	Rissland	Customer Service
Kayoko	Valtorta	Customer Service
Babette	Lamba	Customer Service

Self JOIN

Purwadhika
Startup and Coding School

Self JOIN

A self JOIN is a regular join, but the table is joined with itself.

```
SELECT column_name(s)
FROM table1 T1, table1 T2
WHERE condition;
```

T1 and *T2* are different table aliases for the same table.

Self JOIN

- **New _employees table** SELF JOIN with **Dept_info table** based on **emp_no**.

emp_no	first_name	last_name	gender	hire_date	age
10001	Georgi	Facello	M	1986-06-26	47
10002	Bezalel	Simmel	F	1985-11-21	36
10003	Parto	Bamford	M	1986-08-28	41
10004	Chirstian	Koblick	M	1986-12-01	46
10005	Kyoichi	Maliniak	M	1989-09-12	45
10006	Anneke	Preusig	F	1989-06-02	47
10007	Tzvetan	Zielinski	F	1989-02-10	43
10008	Saniya	Kalloufi	M	1994-09-15	42
10009	Sumant	Peac	F	1985-02-18	48

New_employees tables

emp_no	dept_name
10011	Customer Service
10038	Customer Service
10049	Customer Service
10060	Customer Service
10088	Customer Service
10098	Customer Service
10112	Customer Service
10115	Customer Service
10126	Customer Service
10128	Customer Service

Dept_info tables

Purwadhika
Startup and Coding School

Self JOIN

```
select E.first_name, E.last_name, E.age, D.dept_name
from new_employees E, dept_info D
where E.emp_no = D.emp_no;
```

first_name	last_name	age	dept_name
Mary	Sluis	47	Customer Service
Huan	Lortz	40	Customer Service
Basil	Tramer	39	Customer Service
Breannda	Billingsley	39	Customer Service
Jungsoon	Syrzycki	46	Customer Service
Sreekrishna	Servieres	39	Customer Service
Yuichiro	Swick	37	Customer Service
Chikara	Rissland	36	Customer Service
Kayoko	Valtorta	46	Customer Service
Babette	Lamba	42	Customer Service
Maren	Hutton	41	Customer Service

Reference

- https://www.w3schools.com/sql/sql_join.asp
- https://www.w3schools.com/sql/sql_join_inner.asp
- https://www.w3schools.com/sql/sql_join_left.asp
- https://www.w3schools.com/sql/sql_join_right.asp
- https://www.w3schools.com/sql/sql_join_full.asp
- https://www.w3schools.com/sql/sql_join_self.asp

SESSIONS 1

Access Database Using Python

Data Science Program

Purwadhika
Startup and Coding School

Python MySQL Connector

- MySQL Connector Python enables Python to access MySQL databases
- To connect Python with MySQL, you need to first install **mysql-connector-python**.
- One of the way to install it, is by entering this code on your command prompt/terminal:

```
pip install mysql-connector-python
```

```
C:\Users\Your Name\AppData\Local\Programs\Python\Python36-32\Scripts>python -m pip install mysql-connector-python
```

- Full documentation link: https://dev.mysql.com/doc/connector-python/en/connector-python-introduction.html

Purwadhika
Startup and Coding School

Python MySQL Connector

- Open your VSCode or Jupyter Lab
- Try this code to connect your databases**.** This code open ‘world’ databases

Import Libraries

[1]:
```
import mysql.connector
import pandas as pd
```

Create Connection

[2]:
```
mydb = mysql.connector.connect(
    host = 'localhost',
    user = 'root',
    passwd = 'YourPassword',
    database = 'world'
)
```

Purwadhika
Startup and Coding School

Python MySQL Connector

Access **world** database and open **city** table

Access to Database (1st Method)

[4]:
```
# First Method

# Create access to database
mycursor = mydb.cursor()

# Write query
query = 'select * from city'

# Execute query
mycursor.execute(query)

# save the result in 'result' variable
result = mycursor.fetchall()

# Convert to dataframe and open it
df = pd.DataFrame(result, columns = mycursor.column_names)
df.head(5)
```

[4]:

	ID	Name	CountryCode	District	Population
0	1	Kabul	AFG	Kabol	1780000
1	2	Qandahar	AFG	Qandahar	237500
2	3	Herat	AFG	Herat	186800
3	4	Mazar-e-Sharif	AFG	Balkh	127800
4	5	Amsterdam	NLD	Noord-Holland	731200

Purwadhika
Startup and Coding School

Python MySQL Connector

- Create function to open database and table.
- All you have to do is write query inside function.

Access to Database (2nd Method)

[5]:
```
#Second Method

mycursor = mydb.cursor()

# create function
def sql_df(yourQuery):
    mycursor.execute(yourQuery)
    myresult = mycursor.fetchall()
    df = pd.DataFrame(myresult, columns = mycursor.column_names)
    return df
```

[6]:
```
# trying function
sql_df(
    '''
    select * from city limit 5
    '''
)
```

[6]:

	ID	Name	CountryCode	District	Population
0	1	Kabul	AFG	Kabol	1780000
1	2	Qandahar	AFG	Qandahar	237500
2	3	Herat	AFG	Herat	186800
3	4	Mazar-e-Sharif	AFG	Balkh	127800
4	5	Amsterdam	NLD	Noord-Holland	731200

Purwadhika
Startup and Coding School

Reference

- https://www.w3schools.com/python/python_mysql_getstarted.asp
- https://dev.mysql.com/doc/connector-python/en/connector-python-introduction.html

Purwadhika
Startup and Coding School