Axia Africa

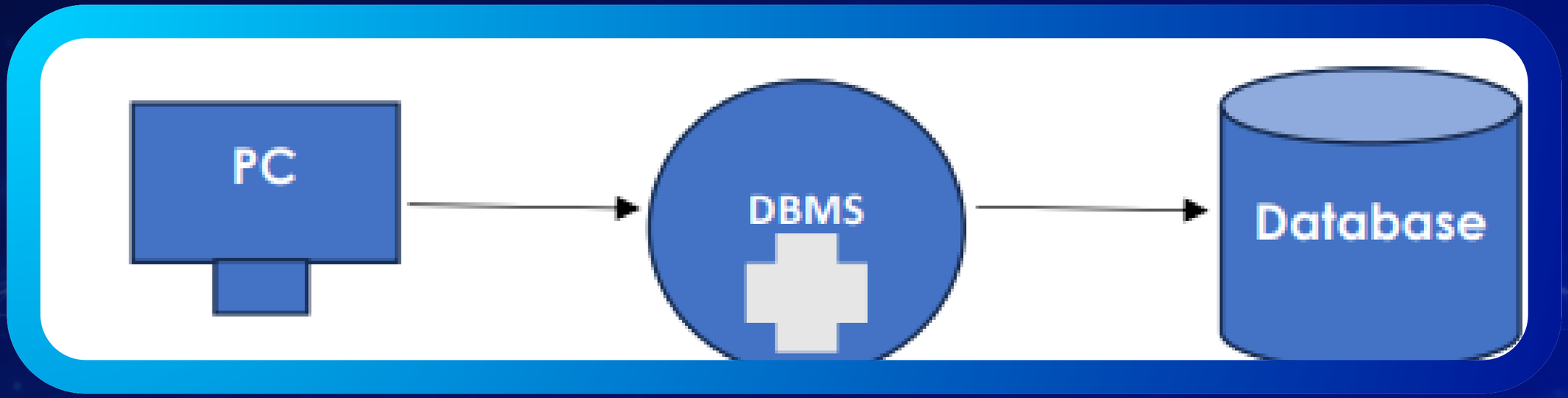
Data Analysis

SQL

CONTENT

INTRODUCTION TO DATABASES AND SQL

A database is an organized collection of related tables which comprises or columns(fields) and rows(record). To manage these databases, DBMS (Database Management System) are used. DBMS is software used to create, manage, and organize databases.



Types of DBMS

- Relational Database
- Non-Relational Database

Our focus for this course would be on relational database.

What is RDBMS?

- RDBMS (Relational Database Management System) - is a DBMS based on the concept of tables (also called relations).

- Data is organized into tables (also known as relations) with rows (records) and columns (attributes). E.g. - MySQL, PostgreSQL, Oracle, MSSQL, Azure Database Studio etc.

Download SQL

What is SQL?
SQL is Structured Query Language – used to store, manipulate, and retrieve data from RDBMS. (It is not a database; it is a language used to interact with database)

We use SQL for CRUD Operations:

- **CREATE – To create databases, tables, insert tuples in tables etc**
- **READ – To read data present in the database.**
- **UPDATE – Modify already inserted data.**
- **DELETE – Delete database, table or specific data point/tuple/row or multiple rows.**

Structured Query Language SQL
(Structured Query Language) is a language that is used to communicate with the database to get needed information and this is done with the use of a database management software (DBMS).

NB: - SQL is not case sensitive, but it is best to write SQL keywords in upper case and others in lower case.
- Every SQL statement should end with a semi colon.

SQL Data Types

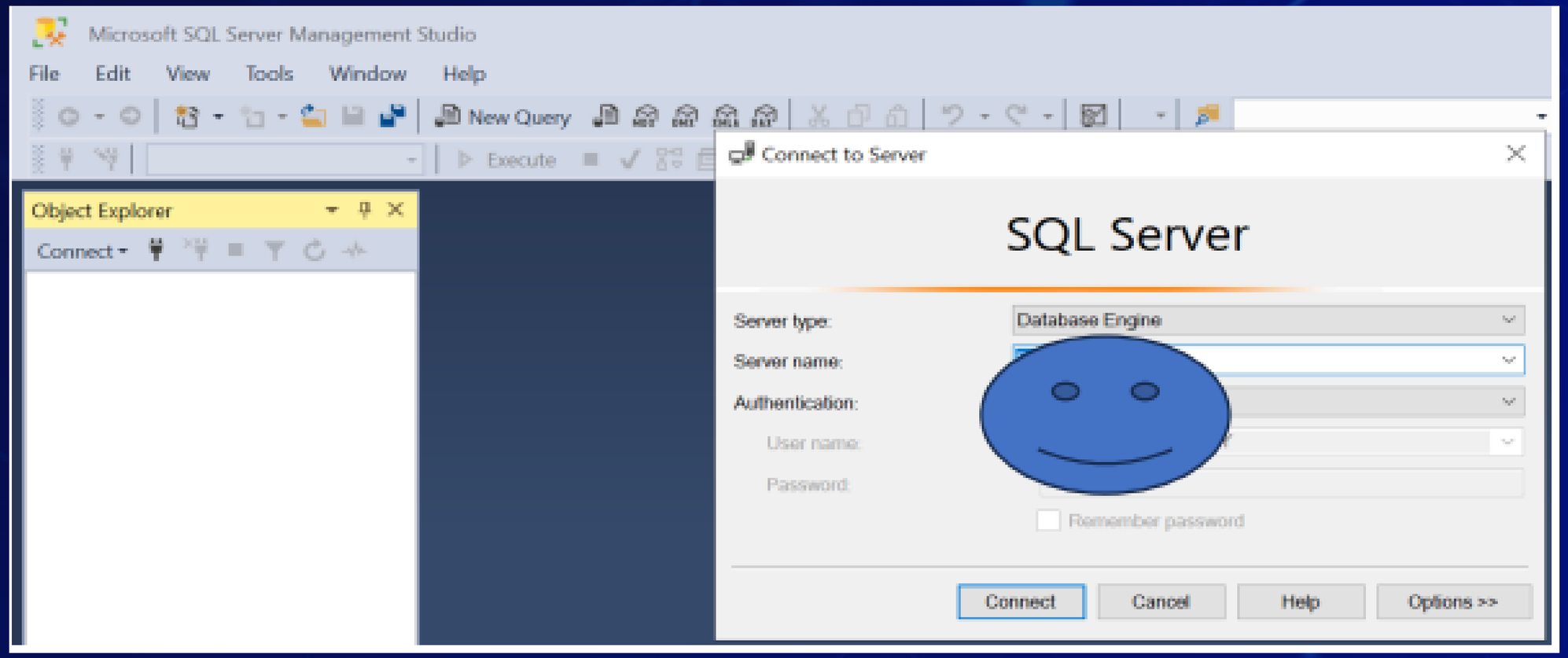
In SQL, data types define the kind of data that can be stored in a column or variable. To See all data types of MYSQL, visit

https://dev.mysql.com/doc/refman/8.0/en/data-types.html

Here are the frequently used SQL data types:

DATATYPE	DESCRIPTION	USAGE
CHAR	string(0-255), can store characters of fixed length	CHAR(50)
VARCHAR	string(0-255), can store characters up to given length	VARCHAR(50)
BLOB	string(0-65535), can store binary large object	BLOB(1000)
INT	integer(-2,147,483,648 to 2,147,483,647)	INT
TINYINT	integer(-128 to 127)	TINYINT
BIGINT	integer(-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807)	BIGINT
BIT	can store x-bit values. x can range from 1 to 64	BIT(2)
FLOAT	Decimal number - with precision to 23 digits	FLOAT
DOUBLE	Decimal number - with 24 to 53 digits	DOUBLE
BOOLEAN	Boolean values 0 or 1	BOOLEAN
DATE	date in format of YYYY-MM-DD ranging from 1000-01-01 to 9999-12-31	DATE
TIME	HH:MM:SS	TIME
YEAR	year in 4 digits format ranging from 1901 to 2155	YEAR

An important point t note is that CHAR is for fixed length & VARCHAR is for variable length strings. Generally, VARCHAR is better as it only occupies necessary memory & works more efficiently.


Microsoft SQL Server Management Studio
File
Edit
View
Tools
Window
Help
New Query
Execute
Object Explorer
Connect
Connect to Server
SQL Server
Server type:
Database Engine
Server name:
Authentication:
User name:
Password:
Remember password
Connect
Cancel
Help
Options >>

TYPES OF SQL COMMAND

1. Data Definition Language (DDL): CREATE, ALTER, DROP
2. Data Manipulation Language (DML): INSERT, UPDATE, DELETE

3. Data Query Language: SELECT

4. Transaction Control Language (TCL): Commit, Rollback

1. Data Definition Language (DDL)

Data Definition Language (DDL) is a subset of SQL (Structured Query Language) responsible for defining and managing the structure of databases and their objects. DDL commands enable you to create, modify, and delete database objects like tables, indexes, constraints, and more. Key DDL Commands are:

CREATE TABLE:

- Used to create a new table in the database.
- Specifies the table name, column names, data types, constraints, and more.

Example:
CREATE TABLE employees (id INT PRIMARY KEY, name VARCHAR(50), salary DECIMAL (10, 2)).

ALTER TABLE:

- Used to modify the structure of an existing table.
- You can add, modify, or drop columns, constraints, and more.
- Example: ALTER TABLE employees ADD COLUMN email VARCHAR (100);

DROP TABLE:

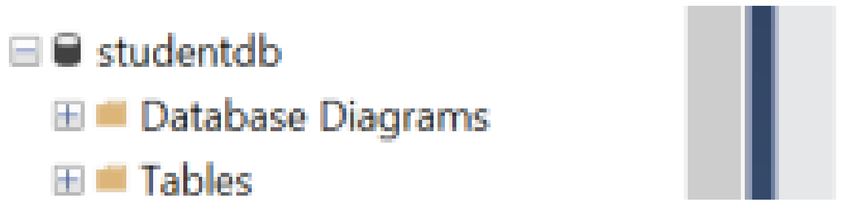
- Used to delete an existing table along with its data and structure.
- Example: DROP TABLE employees;

Creating and Exploring Databases

To create database in SQL, we use this query

```
CREATE database studentdb;
```

In simple terms, CREATE simply means calling something into existence.

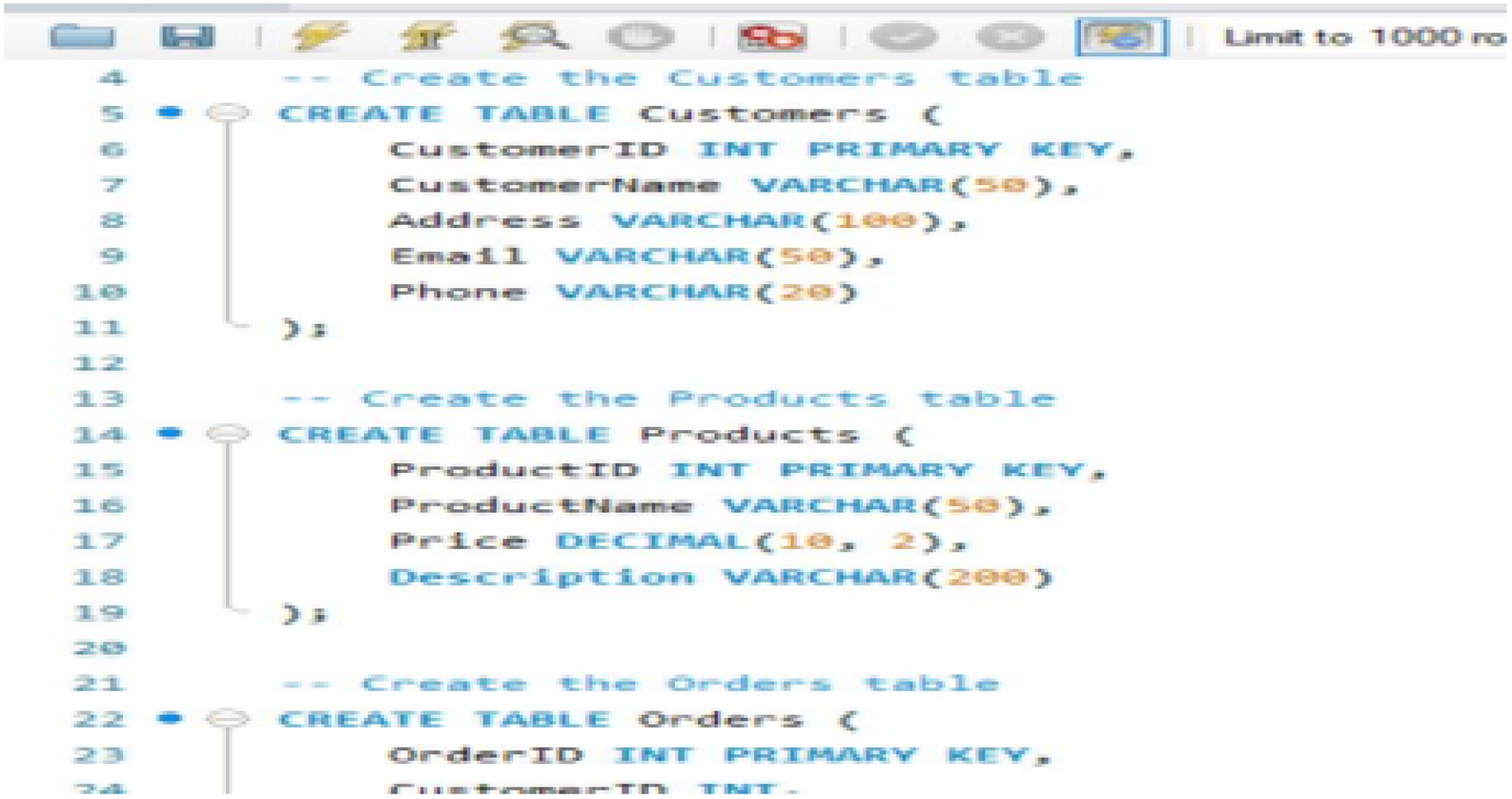



```
13
14  CREATE database studentdb;
15
```

```
-- SQL with prince--
CREATE database hm_stores;
```

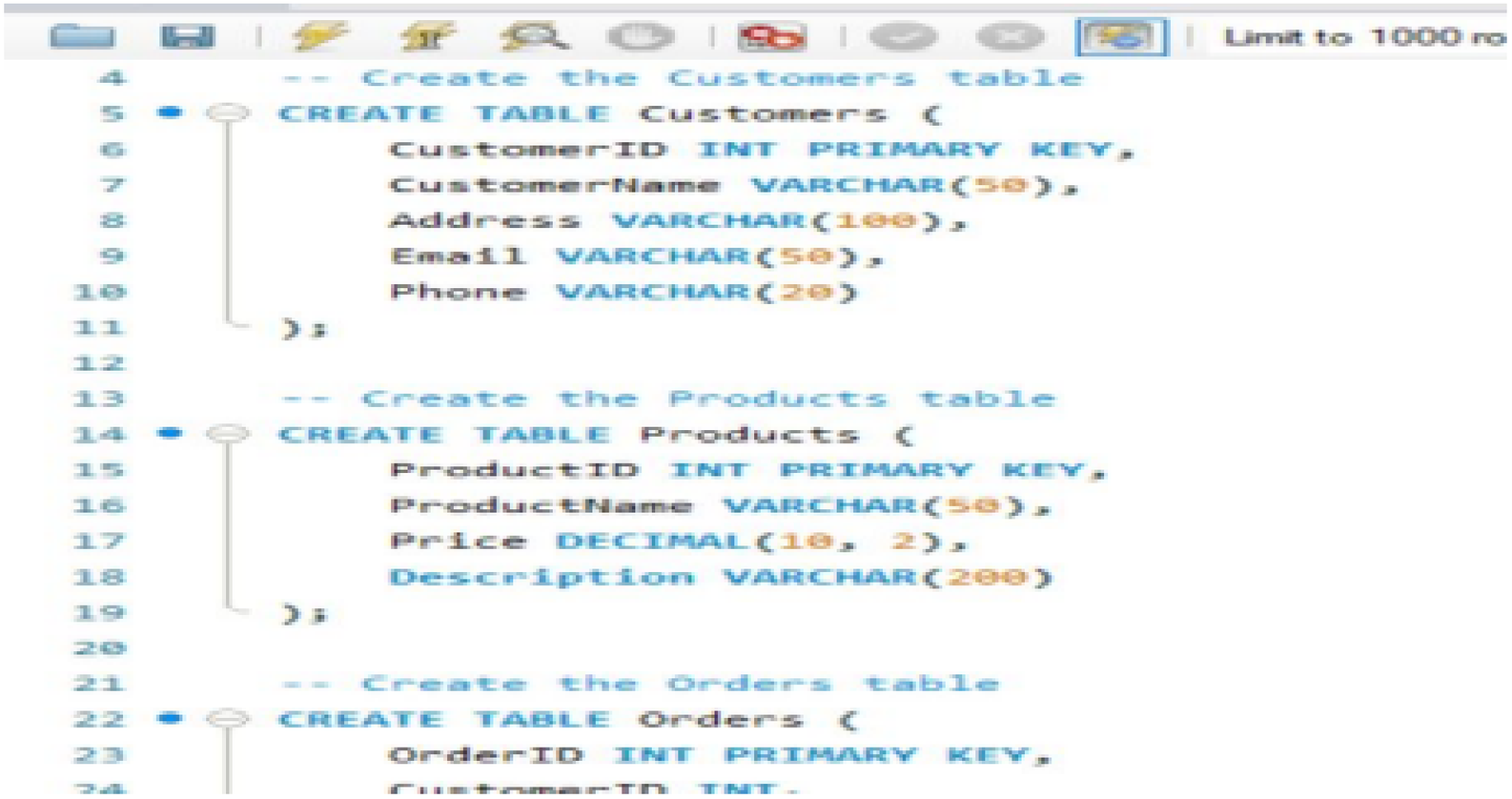
--CREATING A TABLE AND ADD RECORDS TO TABLES IN SQL

In this example we created 3 different tables for our hm_stores database, see image below



--CREATING A TABLE AND ADD RECORDS TO TABLES IN SQL

In this example we created 3 different tables for our hm_stores database, see image below



In this example, the Customers table has columns for CustomerID, CustomerName, Address, Email, and Phone. The Products table includes columns for ProductID, ProductName, Price, and Description. The Orders table contains columns for OrderID, CustomerID, ProductID, OrderDate, and Quantity. The CustomerID and ProductID columns in the Orders table are foreign keys referencing the corresponding primary keys in the Customers and Products tables, respectively.

THE SQL INSERT INTO COMMAND

This command is used to insert data into a table.

INSERT INTO TABLENAME (Column1, Column2, ...)
VALUES(Value1, Value2,....);

```
-- Insert sample data into the Customers table
INSERT INTO Customers (CustomerID, CustomerName, Address, Email, Phone)
VALUES (1, 'John Smith', '123 Main St', 'john@hmstore.com', '123-456-7890'),
       (2, 'Jane Doe', '456 Elm St', 'jane@hmstore.com', '987-654-3210'),
       (3, 'Michael Johnson', '789 Oak St', 'michael@hmstore.com', '555-123-4567'),
       (4, 'Emily Wilson', '321 Pine St', 'emily@hmstore.com', '888-999-0000'),
       (5, 'David Brown', '567 Maple St', 'david@hmstore.com', '444-333-2222'),
       (6, 'Sarah Davis', '901 Cedar St', 'sarah@hmstore.com', '777-888-9999'),
       (7, 'Christopher Lee', '234 Birch St', 'chris@hmstore.com', '222-444-6666'),
       (8, 'Jennifer Taylor', '890 Walnut St', 'jennifer@hmstore.com', '111-222-3333'),
       (9, 'Matthew Wilson', '456 Oak St', 'matthew@hmstore.com', '666-777-8888'),
       (10, 'Olivia Thompson', '678 Elm St', 'olivia@hmstore.com', '999-888-7777');
```

Find the code below;

```
-- Insert sample data into the Customers table
INSERT INTO Customers (CustomerID, CustomerName, Address, Email, Phone)
VALUES (1, 'John Smith', '123 Main St', 'john@hmstore.com', '123-456-7890'),
    (2, 'Jane Doe', '456 Elm St', 'jane@hmstore.com', '987-654-3210'),
    (3, 'Michael Johnson', '789 Oak St', 'michael@hmstore.com', '555-123-4567'),
    (4, 'Emily Wilson', '321 Pine St', 'emily@hmstore.com', '888-999-0000'),
    (5, 'David Brown', '567 Maple St', 'david@hmstore.com', '444-333-2222'),
    (6, 'Sarah Davis', '901 Cedar St', 'sarah@hmstore.com', '777-888-9999'),
    (7, 'Christopher Lee', '234 Birch St', 'chris@hmstore.com', '222-444-6666'),
    (8, 'Jennifer Taylor', '890 Walnut St', 'jennifer@hmstore.com', '111-222-3333'),
    (9, 'Matthew Wilson', '456 Oak St', 'matthew@hmstore.com', '666-777-8888'),
    (10, 'Olivia Thompson', '678 Elm St', 'olivia@hmstore.com', '999-888-7777');
```

```
-- Insert records into the Products table
INSERT INTO Products (ProductID, ProductName, Price, Description)
VALUES (1, 'T-shirt', 19.99, 'Cotton t-shirt'),
    (2, 'Jeans', 59.99, 'Blue denim jeans'),
    (3, 'Sneakers', 79.99, 'Running shoes'),
    (4, 'Dress', 99.99, 'Evening dress'),
    (5, 'Watch', 149.99, 'Wristwatch'),
    (6, 'Backpack', 39.99, 'Canvas backpack'),
    (7, 'Headphones', 129.99, 'Wireless headphones'),
    (8, 'Sunglasses', 89.99, 'Polarized sunglasses'),
    (9, 'Laptop', 999.99, '15-inch laptop'),
    (10, 'Smartphone', 699.99, 'Android smartphone');
```

```
-- Insert records into the Orders table
INSERT INTO Orders (OrderID, CustomerID, ProductID, OrderDate, Quantity)
VALUES (1, 1, 3, '2023-06-01', 2),
       (2, 2, 5, '2023-06-02', 1),
       (3, 3, 1, '2023-06-03', 3),
       (4, 4, 2, '2023-06-04', 1),
       (5, 5, 9, '2023-06-05', 1),
       (6, 6, 4, '2023-06-06', 2),
       (7, 7, 8, '2023-06-07', 1),
       (8, 8, 7, '2023-06-08', 2),
       (9, 9, 6, '2023-06-09', 1),
       (10, 10, 10, '2023-06-10', 1);
```

N/B:
- **The row of a database is known as record or a Tuple.**
- **The column of a database is known as an attribute.**

Assignment.

Create a database “AxiaSchool”, This database should contain two(2) tables known as “coursesTb” & “StudentsTb”. Both tables should hold 20records each.

CourseID	Course	Duration
DA001	Data Analytics	3Months

StudentID	FirstName	LastName	State	Country	CourseID

THE SQL SELECT COMMAND

The Select command is used to retrieve data from the database. The data returned is stored in a result table, called the result-set.

```
SELECT Column1, column2, column3

From TableName;
```

```
1   SELECT  [Id]
2         ,[FirstName]
3         ,[LastName]
4         ,[City]
5         ,[Country]
6         ,[Phone]
7     FROM [AxiaStores].[dbo].[Customer];
```

Results | Messages

	Id	FirstName	LastName	City	Country	Phone
1	1	Maria	Anders	Berlin	Germany	030-0074321
2	2	Ana	Trujillo	México D.F.	Mexico	(5) 555-4729
3	3	Antonio	Moreno	México D.F.	Mexico	(5) 555-3932
4	4	Thomas	Hardy	London	UK	(171) 555-7788
5	5	Christina	Berglund	Luleå	Sweden	0921-12 34 65
6	6	Hanna	Moos	Mannheim	Germany	0621-08460
7	7	Frédérique	Citeaux	Strasbourg	France	88.60.15.31
8	8	Martín	Sommer	Madrid	Spain	(91) 555 22 82
9	9	Laurence	Lebihan	Marseille	France	91.24.45.40
10	10	Elizabeth	Lincoln	Tsawassen	Canada	(604) 555-4729
11	11	Victoria	Ashworth	London	UK	(171) 555-1212
12	12	Patricio	Simpson	Buenos Aires	Argentina	(1) 135-5555
13	13	Francisco	Chang	México D.F.	Mexico	(5) 555-3392

You will get to use the SELECT anytime you need to retrieve data from the database.

SELECT * (This * means All; so, you want to select all records)

FROM (This indicates the table we want to retrieve information from)

WHERE We filter record, we want to retrieve from the database based on a certain condition.

GROUP BY (This is often used with aggregate function to group similar values into summary rows)

ORDER BY Is used to sort data either by ASC – Ascending or DESC – Descending order.
HAVING Is used to filter groups.
JOIN Is used to retrieve data from multiple tables based on a common field.
LIMIT/TOP Is used to specify the number of rows to be returned.

CONSTRAINTS IN SQL

There are certain rules or principles that helps us protect and secure the integrity of our data in a table, this is what we refer to as constraints in SQL. SQL Constraints are rules and restrictions applied on the data in a table.

Here are some examples of such constraints.

- **NOT NULL:** The value in a column cannot be Null.
- **DISTINCT:** value cannot be the same in a column.
- **PRIMARY KEY:** This is used to uniquely identify a row.
- **FOREIGN KEY:** This references a row in another table.

Assignment

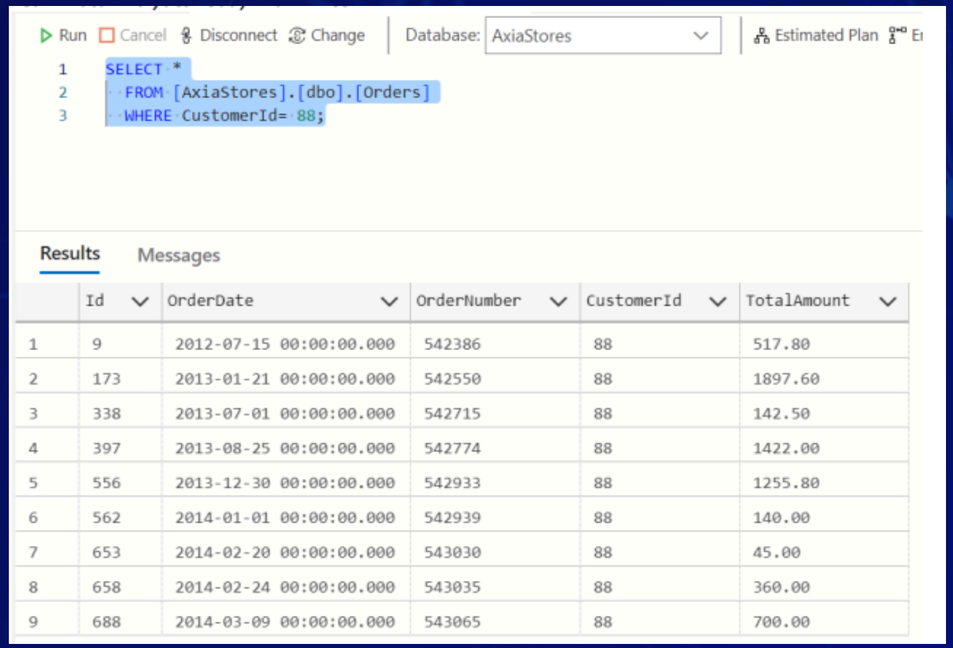
1. List and briefly explain the SQL query order of execution.

THE WHERE OPERATION IN SQL

The WHERE clause is used to filter records.

Syntax: SELECT column1, column2, ... FROM table_name WHERE condition;

Ex: SELECT * FROM Customers WHERE Country='Mexico';



	Id	OrderDate	OrderNumber	CustomerId	TotalAmount
1	9	2012-07-15 00:00:00.000	542386	88	517.80
2	173	2013-01-21 00:00:00.000	542550	88	1897.60
3	338	2013-07-01 00:00:00.000	542715	88	142.50
4	397	2013-08-25 00:00:00.000	542774	88	1422.00
5	556	2013-12-30 00:00:00.000	542933	88	1255.80
6	562	2014-01-01 00:00:00.000	542939	88	140.00
7	653	2014-02-20 00:00:00.000	543030	88	45.00
8	658	2014-02-24 00:00:00.000	543035	88	360.00
9	688	2014-03-09 00:00:00.000	543065	88	700.00

Operators used in WHERE are:
= : Equal
> : Greater than
< : Less than
>= : Greater than or equal
<= : Less than or equal
<> : Not equal.
Note: In some versions of SQL this operator may be written as !=

THE AND, OR and NOT OPERATION IN SQL:

- The WHERE clause can be combined with AND, OR, and NOT operators.
- The AND and OR operators are used to filter records based on more than one condition.
- The AND operator displays a record if all the conditions separated by AND are TRUE.
- The OR operator displays a record if any of the conditions separated by OR is TRUE.
- The NOT operator displays a record if the condition(s) is NOT TRUE.

Syntax:

SELECT column1, column2, ... FROM table_name WHERE condition1 AND condition2 AND condition3 ...;

SELECT column1, column2, ... FROM table_name WHERE condition1 OR condition2 OR condition3 ...;

SELECT column1, column2, ... FROM table_name WHERE NOT condition;

Example:

```
SELECT * FROM Customers WHERE Country='India' AND City='Japan';

SELECT * FROM Customers WHERE Country='America' AND (City='India' OR
City='Korea');
```

DISTINCT IN SQL

The distinct removes duplicate rows from query results.

Syntax: SELECT DISTINCT column1, column2 FROM table_name;

THE LIKE OPERATION:

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 sign (_) represents one, single character.

Example: SELECT * FROM employees WHERE first_name LIKE 'J%';

WHERE CustomerName LIKE 'a%'

- Finds any values that start with "a"

WHERE CustomerName LIKE '%a'

- Finds any values that end with "a"

WHERE CustomerName LIKE '%or%'

- Finds any values that have "or" in any position

WHERE CustomerName LIKE '_r%'

- Finds any values that have "r" in the second position

WHERE CustomerName LIKE 'a_%'

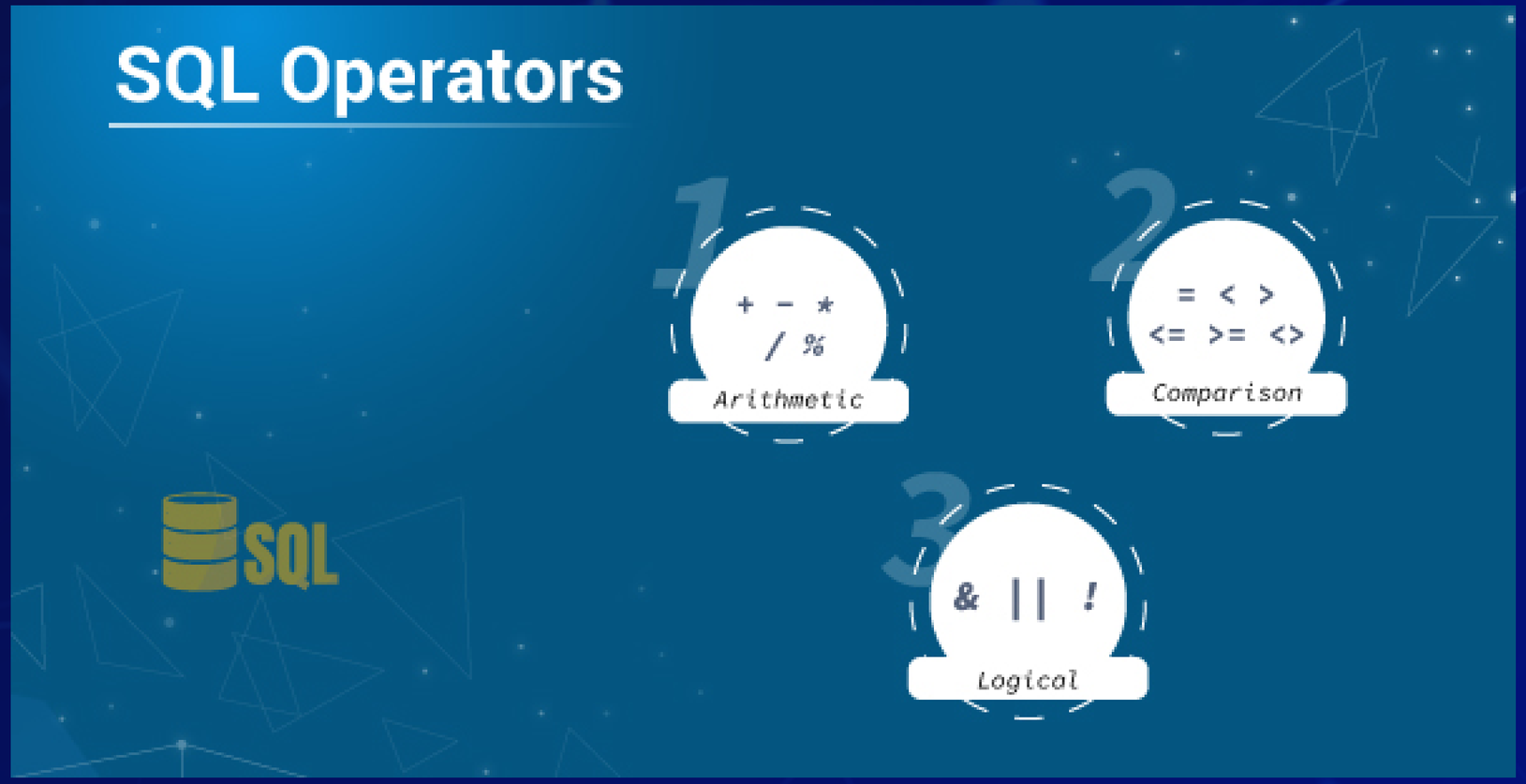
- Finds any values that start with "a" and are at least 2 characters in length

WHERE CustomerName LIKE 'a__%'

- Finds any values that start with "a" and are at least 3 characters in length

WHERE ContactName LIKE 'a%o'

- Finds any values that start with "a" and ends with "o"


SQL Operators
1
+ - *
/ %
Arithmetic
2
= < >
<= >= <>
Comparison
3
& || !
Logical
SQL

IN OPERATION:

Filters results based on a list of values in the WHERE clause.

Example: SELECT * FROM products WHERE category_id IN (1, 2, 3);

BETWEEN OPERATION

Filters results within a specified range in the WHERE clause.

Example: SELECT * FROM orders WHERE order_date BETWEEN '2023-01-01' AND '2023-06-30';

IS NULL OPERATION

This checks for NULL values in the WHERE clause.

Example: SELECT * FROM customers WHERE email IS NULL;

THE AS OPERATION

In SQL, the AS is also called aliases. This renames columns or expressions in query results.

Example: SELECT first_name AS "First Name", last_name AS "Last Name" FROM employees;

ORDER BY

The ORDER BY clause allows you to sort the result set of a query based on one or more columns.

Basic Syntax:

The ORDER BY clause is used after the SELECT statement to sort query results.

Syntax: SELECT column1, column2 FROM table_name ORDER BY column1 [ASC | DESC];

Ascending and Descending Order:

By default, the ORDER BY clause sorts in ascending order (smallest to largest).

You can explicitly specify descending order using the DESC keyword.

Example: SELECT product_name, price FROM products ORDER BY price DESC;

Sorting by Multiple Columns:

You can sort by multiple columns by listing them sequentially in the ORDER BY clause. Rows are first sorted based on the first column, and for rows with equal values, subsequent columns are used for further sorting.

Example: SELECT first_name, last_name FROM employees ORDER BY last_name, first_name;

Sorting by Expressions:

It's possible to sort by calculated expressions, not just column values.

Example: SELECT product_name, price, price * 1.1 AS discounted_price FROM products ORDER BY discounted_price;

Sorting NULL Values:

By default, NULL values are considered the smallest in ascending order and the largest in descending order. You can control the sorting behaviour of NULL values using the NULLS FIRST or NULLS LAST options.

Example: SELECT column_name FROM table_name ORDER BY column_name NULLS LAST;

Sorting by Position:

Instead of specifying column names, you can sort by column positions in the ORDER BY clause.

Example: SELECT product_name, price FROM products
ORDER BY 2 DESC, 1 ASC;

GROUP BY: The GROUP BY clause in SQL is used to group rows from a table based on one or more columns.

Syntax:

The GROUP BY clause follows the SELECT statement and is used to group rows based on specified columns.

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

Aggregation Functions: Aggregation functions (e.g., COUNT, SUM, AVG, MAX, MIN) are often used with GROUP BY to calculate values for each group.

Example: SELECT department, AVG(salary) FROM employees GROUP BY department;

Grouping by Multiple Columns:

You can group by multiple columns by listing them in the GROUP BY clause. This creates a hierarchical grouping based on the specified columns.

Example: SELECT department, gender, AVG(salary) FROM employees GROUP BY department, gender;

HAVING Clause:

- The HAVING clause is used with GROUP BY to filter groups based on aggregate function results.
- It's like the WHERE clause but operates on grouped data.

Example: SELECT department, AVG(salary) FROM employees GROUP BY department HAVING AVG(salary) > 50000;

Combining GROUP BY and ORDER BY: You can use both GROUP BY and ORDER BY in the same query to control the order of grouped results.

Example: SELECT department, COUNT(*) FROM employees GROUP BY department ORDER BY COUNT(*) DESC;

SQL AGGREGATE FUNCTIONS

These are used to perform calculations on groups of rows or entire result sets. They provide insights into data by summarising and processing information.

Some Common Aggregate Functions:

- **COUNT():**

Counts the number of rows in a group or result set.

- **SUM():**

Calculates the sum of numeric values in a group or result set.

- **AVG():**

Computes the average of numeric values in a group or result set.

- **MAX():**

Finds the maximum value in a group or result set.

- **MIN():**

Retrieves the minimum value in a group or result set.

Advanced SQL Queries for Analysis.

- **Introduction to SQL JOINs and UNIONs**

SQL JOIN SQL statement is used to combine data or rows from two or more tables based on a common field between them. Different types of Joins are as follows:

- INNER JOIN
- LEFT JOIN
- RIGHT JOIN
- FULL

JOIN combines columns horizontally from multiple tables based on join conditions, allowing you to retrieve related data.

- **INNER JOIN**

An inner join combines data from two or more tables based on a specified condition, known as the join condition.

The result of an inner join includes only the rows where the join condition is met in all participating tables. It essentially filters out non-matching rows and returns only the rows that have matching values in both tables.

SQL AGGREGATE FUNCTIONS

These are used to perform calculations on groups of rows or entire result sets. They provide insights into data by summarising and processing information.

Some Common Aggregate Functions:

- **COUNT():**

Counts the number of rows in a group or result set.

- **SUM():**

Calculates the sum of numeric values in a group or result set.

- **AVG():**

Computes the average of numeric values in a group or result set.

- **MAX():**

Finds the maximum value in a group or result set.

- **MIN():**

Retrieves the minimum value in a group or result set.

Advanced SQL Queries for Analysis.

- **Introduction to SQL JOINs and UNIONs**

SQL JOIN SQL statement is used to combine data or rows from two or more tables based on a common field between them. Different types of Joins are as follows:

- INNER JOIN
- LEFT JOIN
- RIGHT JOIN
- FULL

JOIN combines columns horizontally from multiple tables based on join conditions, allowing you to retrieve related data.

- **INNER JOIN**

An inner join combines data from two or more tables based on a specified condition, known as the join condition.

The result of an inner join includes only the rows where the join condition is met in all participating tables. It essentially filters out non-matching rows and returns only the rows that have matching values in both tables.

Syntax:

```
SELECT columns
FROM table1
INNER JOIN table2
ON table1.column = table2.column;
```

Here:

- columns refer to the specific columns you want to retrieve from the tables.
- table1 and table2 are the names of the tables you are joining.
- column is the common column used to match rows between the tables.
- The ON clause specifies the join condition, where you define how the tables are related.

Example: Consider two tables: Customers and Orders.

Run | Cancel | Disconnect | Change | Database: AxiaStores | Estimated Plan | Enable Actual Plan

```
1  SELECT c.FirstName, c.LastName, c.City,c.Country,o.OrderDate,o.TotalAmount AS Revenue
2  FROM [Orders] o
3  JOIN [Customer] c ON
4  o.CustomerId = c.Id
5  ORDER BY o.TotalAmount DESC;
```

Results | Messages

	FirstName	LastName	City	Country	OrderDate	Revenue
1	Horst	Kloss	Cunewalde	Germany	2014-02-02 00:00:00.000	17250.00
2	Jose	Pavarotti	Boise	USA	2014-04-17 00:00:00.000	16321.90
3	Mario	Pontes	Rio de Janeiro	Brazil	2014-03-27 00:00:00.000	15810.00
4	Lúcia	Carvalho	Sao Paulo	Brazil	2012-12-04 00:00:00.000	12281.20
5	Jean	Fresnière	Montréal	Canada	2013-01-23 00:00:00.000	11493.20
6	Philip	Cramer	Brandenburg	Germany	2014-01-06 00:00:00.000	11490.70
7	Paula	Wilson	Albuquerque	USA	2014-02-16 00:00:00.000	11380.00
8	Jytte	Petersen	Kobenhavn	Denmark	2013-01-16 00:00:00.000	11283.20
9	Patricia	McKenna	Cork	Ireland	2014-02-19 00:00:00.000	10835.24
10	Georg	Pipps	Salzburg	Austria	2012-11-13 00:00:00.000	10741.60
11	Horst	Kloss	Cunewalde	Germany	2013-04-23 00:00:00.000	10588.50
12	Paula	Wilson	Albuquerque	USA	2013-03-19 00:00:00.000	10495.60
13	Horst	Kloss	Cunewalde	Germany	2013-05-19 00:00:00.000	10191.70
14	Horst	Kloss	Cunewalde	Germany	2013-10-03 00:00:00.000	10164.80
15	Karl	Jablonski	Seattle	USA	2014-04-17 00:00:00.000	8902.50
16	Howard	Snyder	Eugene	USA	2014-01-06 00:00:00.000	8891.00
17	Roland	Mendel	Graz	Austria	2013-04-22 00:00:00.000	8623.45

The Syntax Explained:

Syntax:

```
SELECT table1.column1,table1.column2,table2.column1,....
FROM table1
INNER JOIN table2
ON table1.matching_column = table2.matching_column;
table1: First table.
table2: Second table
matching_column: Column common to both the tables.
```

Note: We can also write JOIN instead of INNER JOIN. JOIN is same as INNER JOIN.

```
SELECT CONCAT(c.FirstName, ' ', c.LastName) AS CustomerName, c.City,c.Country,o.OrderDate,o.TotalAmount AS Revenue
FROM [Orders] o
JOIN [Customer] c ON
o.CustomerId = c.Id
ORDER BY o.TotalAmount DESC;
```

Results Messages

	CustomerName	City	Country	OrderDate	Revenue
1	Horst Kloss	Cunewalde	Germany	2014-02-02 00:00:00.000	17250.00
2	Jose Pavarotti	Boise	USA	2014-04-17 00:00:00.000	16321.90
3	Mario Pontes	Rio de Janeiro	Brazil	2014-03-27 00:00:00.000	15810.00
4	Lúcia Carvalho	Sao Paulo	Brazil	2012-12-04 00:00:00.000	12281.20
5	Jean Fresnière	Montréal	Canada	2013-01-23 00:00:00.000	11493.20
6	Philip Cramer	Brandenburg	Germany	2014-01-06 00:00:00.000	11490.70
7	Paula Wilson	Albuquerque	USA	2014-02-16 00:00:00.000	11380.00
8	Jytte Petersen	Kobenhavn	Denmark	2013-01-16 00:00:00.000	11283.20
9	Patricia McKenna	Cork	Ireland	2014-02-19 00:00:00.000	10835.24
10	Georg Pipps	Salzburg	Austria	2012-11-13 00:00:00.000	10741.60
11	Horst Kloss	Cunewalde	Germany	2013-04-23 00:00:00.000	10588.50

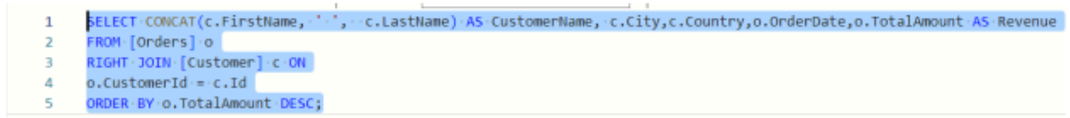
Right Outer Join (Right Join):

A right outer join is like a left outer join, but it returns all rows from the right table and the matching rows from the left table. If there is no match in the left table, the result will still include the right table's row with NULL values in the left table's columns.

	CustomerName	City	Country	OrderDate	Revenue
815	Yoshi Tannamuri	Vancouver	Canada	2013-08-05 00:00:00.000	57.50
816	Giovanni Rovelli	Bergamo	Italy	2013-11-25 00:00:00.000	55.20
817	Dominique Perri…	Paris	France	2013-11-12 00:00:00.000	52.35
818	Paolo Accorti	Torino	Italy	2013-01-22 00:00:00.000	49.80
819	Art Braunschwei…	Lander	USA	2012-08-01 00:00:00.000	48.00
820	Helen Bennett	Cowes	UK	2013-09-18 00:00:00.000	45.00
821	Simon Crowther	London	UK	2014-04-29 00:00:00.000	45.00
822	Annette Roulet	Toulouse	France	2014-04-27 00:00:00.000	45.00
823	Paula Parente	Resende	Brazil	2014-02-20 00:00:00.000	45.00
824	Jose Pavarotti	Boise	USA	2014-01-05 00:00:00.000	40.00
825	Fran Wilson	Portland	USA	2014-02-12 00:00:00.000	36.00
826	Yvonne Moncada	Buenos Aires	Argentina	2014-02-20 00:00:00.000	30.00
827	Pascale Cartrain	Charleroi	Belgium	2013-12-05 00:00:00.000	28.00
828	Maurizio Moroni	Reggio Emilia	Italy	2013-07-02 00:00:00.000	28.00
829	Paolo Accorti	Torino	Italy	2013-12-31 00:00:00.000	18.40
830	Patricio Simpson	Buenos Aires	Argentina	2013-12-17 00:00:00.000	12.50
831	Diego Roel	Madrid	Spain	*NULL*	*NULL*
832	Marie Bertrand	Paris	France	*NULL*	*NULL*

Example: Using the same Customers and Orders tables.

```
SELECT CONCAT(c.FirstName, ' ', c.LastName) AS CustomerName,
c.City,c.Country,o.OrderDate,o.TotalAmount AS Revenue
FROM [Orders] o
RIGHT JOIN [Customer] c ON
o.CustomerId = c.Id
ORDER BY o.TotalAmount DESC;
```



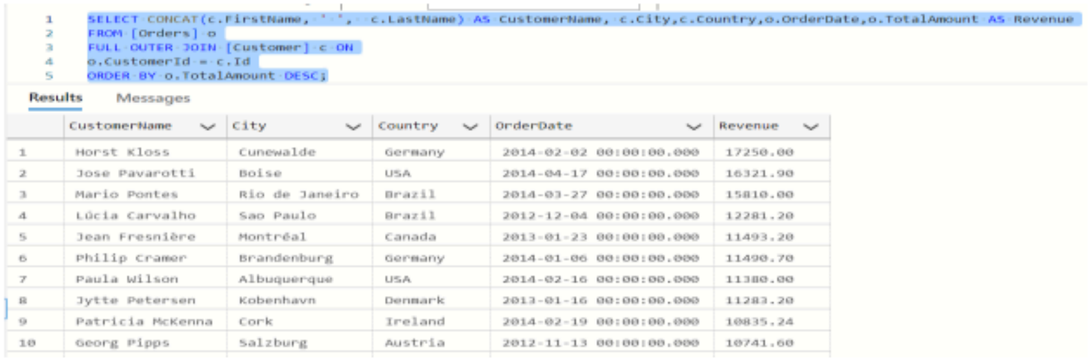
Results | Messages

	CustomerName	City	Country	OrderDate	Revenue
1	Horst Kloss	Cunewalde	Germany	2014-02-02 00:00:00.000	17250.00
2	Jose Pavarotti	Boise	USA	2014-04-17 00:00:00.000	16321.90
3	Mario Pontes	Rio de Janeiro	Brazil	2014-03-27 00:00:00.000	15810.00
4	Lúcia Carvalho	Sao Paulo	Brazil	2012-12-04 00:00:00.000	12281.20
5	Jean Fresnière	Montréal	Canada	2013-01-23 00:00:00.000	11493.20
6	Philip Cramer	Brandenburg	Germany	2014-01-06 00:00:00.000	11490.70
7	Paula Wilson	Albuquerque	USA	2014-02-16 00:00:00.000	11380.00
8	Jytte Petersen	Kobenhavn	Denmark	2013-01-16 00:00:00.000	11283.20
9	Patricia McKenna	Cork	Ireland	2014-02-19 00:00:00.000	10835.24
10	Georg Pipps	Salzburg	Austria	2012-11-13 00:00:00.000	10741.60
11	Horst Kloss	Cunewalde	Germany	2013-04-23 00:00:00.000	10588.50
12	Paula Wilson	Albuquerque	USA	2013-03-19 00:00:00.000	10495.60
13	Horst Kloss	Cunewalde	Germany	2013-05-19 00:00:00.000	10191.70
14	Horst Kloss	Cunewalde	Germany	2013-10-03 00:00:00.000	10164.80
15	Karl Jablonski	Seattle	USA	2014-04-17 00:00:00.000	8902.50
16	Howard Snyder	Eugene	USA	2014-01-06 00:00:00.000	8891.00
17	Roland Mendel	Graz	Austria	2013-04-22 00:00:00.000	8623.45

Full Outer Join (Full Join):

A full outer join returns all rows from both the left and right tables, including matches and non-matches. If there's no match, NULL values appear in columns from the table where there's no corresponding value.

Example: Using the same Customers and Orders tables.



```
SELECT CONCAT(c.FirstName, ' ', c.LastName) AS CustomerName, c.City,c.Country,o.OrderDate,o.TotalAmount AS Revenue
FROM [Orders] o
FULL OUTER JOIN [Customer] c ON
o.CustomerId = c.Id
ORDER BY o.TotalAmount DESC;
```

Results | Messages

	CustomerName	City	Country	OrderDate	Revenue
1	Horst Kloss	Cunewalde	Germany	2014-02-02 00:00:00.000	17250.00
2	Jose Pavarotti	Boise	USA	2014-04-17 00:00:00.000	16321.90
3	Mario Pontes	Rio de Janeiro	Brazil	2014-03-27 00:00:00.000	15810.00
4	Lúcia Carvalho	Sao Paulo	Brazil	2012-12-04 00:00:00.000	12281.20
5	Jean Fresnière	Montréal	Canada	2013-01-23 00:00:00.000	11493.20
6	Philip Cramer	Brandenburg	Germany	2014-01-06 00:00:00.000	11490.70
7	Paula Wilson	Albuquerque	USA	2014-02-16 00:00:00.000	11380.00
8	Jytte Petersen	Kobenhavn	Denmark	2013-01-16 00:00:00.000	11283.20
9	Patricia McKenna	Cork	Ireland	2014-02-19 00:00:00.000	10835.24
10	Georg Pipps	Salzburg	Austria	2012-11-13 00:00:00.000	10741.60

❖ Cross Join

A cross join, also known as a Cartesian product, is a type of join operation in a Database Management System (DBMS) that combines every row from one table with every row from another table.

Unlike other join types, a cross join does not require a specific condition to match rows between the tables. Instead, it generates a result set that contains all possible combinations of rows from both tables.

Cross joins can lead to a large result set, especially when the participating tables have many rows.

Syntax:

```
SELECT columns
FROM table1
CROSS JOIN table2;
```

In this syntax:

- columns refer to the specific columns you want to retrieve from the cross-joined tables.
- table1 and table2 are the names of the tables you want to combine using a cross join.

Example: Consider two tables: Students and Courses we created earlier on in this course.

SET OPERATIONS

Set operations in SQL are used to combine or manipulate the result sets of multiple **SELECT** queries. They allow you to perform operations like those in set theory, such as union, intersection, and difference, on the data retrieved from different tables or queries.

Set operations provide powerful tools for managing and manipulating data, enabling you to analyse and combine information in various ways.

There are four primary set operations in SQL:

- UNION
- INTERSECT
- EXCEPT (or MINUS)
- UNION ALL

Combining data from multiple tables in SQL using UNION

UNION combines rows from different tables vertically, creating a unified result set. UNION is an operator used to combine the result sets of two or more SELECT statements into a single result set.

The SELECT statements involved in a UNION operation must have the same number of columns, and the corresponding columns must have compatible data types.

UNION eliminates duplicate rows from the result set, providing a distinct set of rows.

The UNION operation is useful when you want to merge data from multiple sources or retrieve a consolidated view of data that meets certain criteria.

```
16  -- UNION
17  SELECT Name AS EntityName, 'Class' AS EntityType
18  FROM Production.Product
19  UNION
20  SELECT FirstName + ' ' + LastName AS EntityName, 'Person' AS EntityType
21  FROM Person.Person;
```

100 %

Results | Messages

	EntityName	EntityType
1	HL Mountain Frame - Black, 44	Class
2	LL Road Frame - Red, 62	Class
3	Road-250 Black, 58	Class
4	Seat Tube	Class
5	Sam Abolrous	Person
6	Willie Anand	Person
7	James Young	Person
8	Jaclyn Zheng	Person
9	Claudia Zhou	Person
10	Larry Vazquez	Person
11	Austin White	Person
12	Dylan Williams	Person
13	Seth Wright	Person

Query executed successfully. DATAGUY (15.0 RTM) DATAGUY\DATAG

```
10
11  SELECT FirstName, LastName, 'Customer' AS Type
12  FROM Customer
13  UNION
14  SELECT CompanyName,ContactName, 'Supplier' AS Type
15  FROM Supplier;
16
```

100 %

Results | Messages

	FirstName	LastName	Type
110	Nord-Os...	Sven Petersen	Supplier
111	Norske ...	Beate Vileid	Supplier
112	Pasta B...	Giovanni Giudici	Supplier
113	Pavlova, ...	Ian Devling	Supplier
114	PB Knäc...	Lars Peterson	Supplier
115	Plutzer L...	Martin Bein	Supplier
116	Refresc...	Carlos Diaz	Supplier
117	Specialt...	Peter Wilson	Supplier
118	Svensk ...	Michael Björn	Supplier
119	Tokyo Tr...	Yoshi Nagase	Supplier
120	Zaanse ...	Dirk Luchte	Supplier

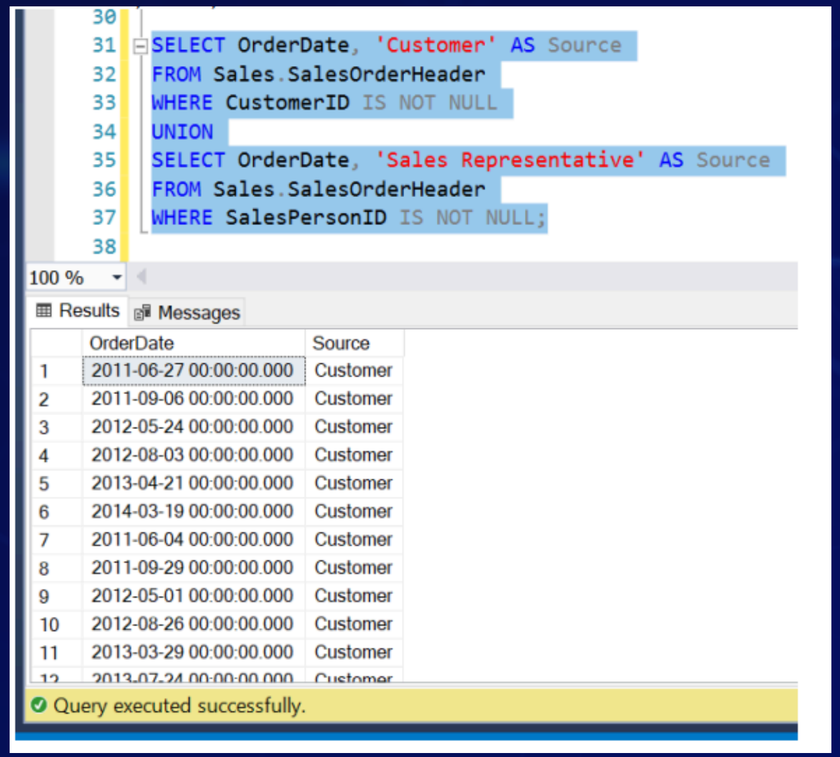
This query combines the FirstName and LastName columns from the Customer table with the CompanyName and ContactName columns from the Suppliers table, and adds a new column 'Type' to indicate whether the record belongs to a customer or a supplier.

```
23
24  SELECT Name
25  FROM Production.ProductCategory
26  UNION
27  SELECT Name
28  FROM Production.ProductSubcategory;
```

100 %

Results | Messages

	Name
1	Accessories
2	Bib-Shorts
3	Bike Racks
4	Bike Stands
5	Bikes
6	Bottles and Cages
7	Bottom Brackets
8	Brakes
9	Caps
10	Chains
11	Cleaners
12	Clothing



```
30
31  SELECT OrderDate, 'Customer' AS Source
32  FROM Sales.SalesOrderHeader
33  WHERE CustomerID IS NOT NULL
34  UNION
35  SELECT OrderDate, 'Sales Representative' AS Source
36  FROM Sales.SalesOrderHeader
37  WHERE SalesPersonID IS NOT NULL;
38
```

100 %

Results | Messages

	OrderDate	Source
1	2011-06-27 00:00:00.000	Customer
2	2011-09-06 00:00:00.000	Customer
3	2012-05-24 00:00:00.000	Customer
4	2012-08-03 00:00:00.000	Customer
5	2013-04-21 00:00:00.000	Customer
6	2014-03-19 00:00:00.000	Customer
7	2011-06-04 00:00:00.000	Customer
8	2011-09-29 00:00:00.000	Customer
9	2012-05-01 00:00:00.000	Customer
10	2012-08-26 00:00:00.000	Customer
11	2013-03-29 00:00:00.000	Customer
12	2013-07-24 00:00:00.000	Customer

Query executed successfully.

INTERSECT:

The INTERSECT operator returns the common rows that exist in the result sets of two or more SELECT queries. It only returns distinct rows that appear in all result sets.

Example: Using the same tables as before.

SELECT CustomerName FROM Customers

INTERSECT

SELECT SupplierName FROM Suppliers;

EXCEPT (or MINUS):

The EXCEPT operator (also known as MINUS in some databases) returns the distinct rows that are present in the result set of the first SELECT query but not in the result set of the second SELECT query.

Example: Using the same tables as before.

```
SELECT CustomerName FROM Customers
EXCEPT
SELECT SupplierName FROM Suppliers;
```

UNION ALL:

The UNION ALL operator performs the same function as the UNION operator but does not remove duplicates from the result set. It simply concatenates all rows from the different result sets.

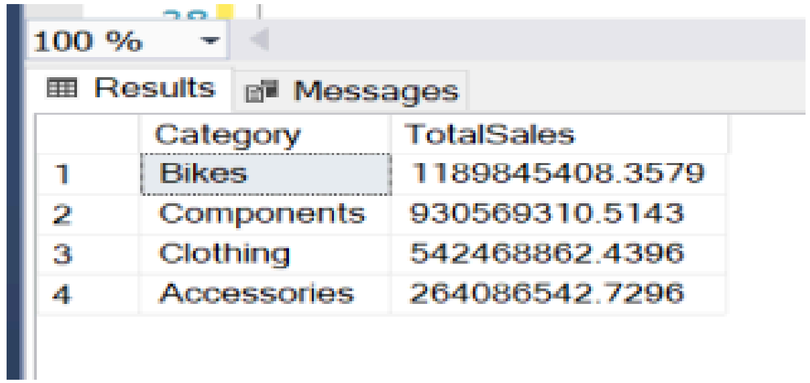
Example: Using the same tables as before.

```
SELECT CustomerName FROM Customers
UNION ALL
SELECT SupplierName FROM Suppliers;
```

- Exploring aggregate functions in SQL (SUM, COUNT, AVG, GROUP BY, etc)

1. SUM

```
-- SUM - Total Sales Amount by Product Category and sort in Descending Order:
SELECT pc.Name AS Category, SUM(soh.TotalDue) AS TotalSales
FROM Sales.SalesOrderHeader AS soh
JOIN Sales.SalesOrderDetail AS sod ON soh.SalesOrderID = sod.SalesOrderID
JOIN Production.Product AS p ON sod.ProductID = p.ProductID
JOIN Production.ProductSubcategory AS psc ON p.ProductSubcategoryID =
psc.ProductSubcategoryID
JOIN Production.ProductCategory AS pc ON psc.ProductCategoryID = pc.ProductCategoryID
GROUP BY pc.Name
ORDER BY TotalSales DESC;
```



100 %

Results | Messages

	Category	TotalSales
1	Bikes	1189845408.3579
2	Components	930569310.5143
3	Clothing	542468862.4396
4	Accessories	264086542.7296

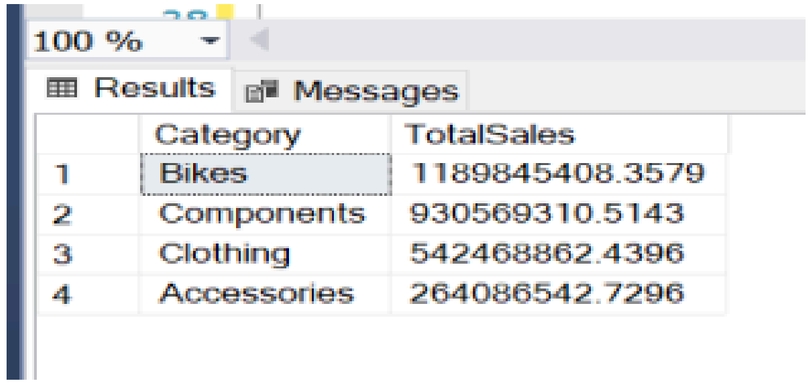
2. COUNT:

```
-- COUNT - Number of Employees by Job Title
SELECT JobTitle, COUNT(*) AS NumEmployees
FROM HumanResources.Employee
GROUP BY JobTitle;
```

- Exploring aggregate functions in SQL (SUM, COUNT, AVG, GROUP BY, etc)

1. SUM

```
-- SUM - Total Sales Amount by Product Category and sort in Descending Order:
SELECT pc.Name AS Category, SUM(soh.TotalDue) AS TotalSales
FROM Sales.SalesOrderHeader AS soh
JOIN Sales.SalesOrderDetail AS sod ON soh.SalesOrderID = sod.SalesOrderID
JOIN Production.Product AS p ON sod.ProductID = p.ProductID
JOIN Production.ProductSubcategory AS psc ON p.ProductSubcategoryID =
psc.ProductSubcategoryID
JOIN Production.ProductCategory AS pc ON psc.ProductCategoryID = pc.ProductCategoryID
GROUP BY pc.Name
ORDER BY TotalSales DESC;
```



100 %

Results | Messages

	Category	TotalSales
1	Bikes	1189845408.3579
2	Components	930569310.5143
3	Clothing	542468862.4396
4	Accessories	264086542.7296

2. COUNT:

```
-- COUNT - Number of Employees by Job Title
SELECT JobTitle, COUNT(*) AS NumEmployees
FROM HumanResources.Employee
GROUP BY JobTitle;
```

```sql
-- COUNT - Number of Employees by Job Title
SELECT JobTitle, COUNT(*) AS NumEmployees
FROM HumanResources.Employee
GROUP BY JobTitle;
```

100 %

Results | Messages

	JobTitle	NumEmployees
1	Accountant	2
2	Accounts Manager	1
3	Accounts Payable Specialist	2
4	Accounts Receivable Specialist	3
5	Application Specialist	4
6	Assistant to the Chief Financial Officer	1
7	Benefits Specialist	1
8	Buyer	9
9	Chief Executive Officer	1
10	Chief Financial Officer	1
11	Control Specialist	2
12	Database Administrator	2

Query executed successfully.

This query counts the number of employees (COUNT) for each unique job title in the HumanResources.Employee table. The results are grouped by job title.

Quiz: Try to sort it in descending order.

3. Avg Price calculation: To calculate the average list price (AVG) for each product subcategory by joining the necessary tables and grouping the results based on the subcategory name and then sort the result in descending order..

```sql
--AVG - Average List Price by Product Subcategory:
SELECT psc.Name AS Subcategory, ROUND(AVG(p.ListPrice), 2) AS AvgListPrice
FROM Production.Product AS p
JOIN Production.ProductSubcategory AS psc ON p.ProductSubcategoryID = psc.ProductSubcategoryID
GROUP BY psc.Name
ORDER BY AvgListPrice DESC;
```

100 %

Results | Messages

	Subcategory	AvgListPrice
1	Mountain Bikes	1683.37
2	Road Bikes	1597.45
3	Touring Bikes	1425.25
4	Road Frames	780.04
5	Mountain Frames	678.25
6	Touring Frames	631.42
7	Cranksets	278.99
8	Wheels	228.93
9	Forks	184.40
10	Bike Stands	159.00
11	Panniers	125.00
12	Bike Racks	120.00

Query executed successfully. DATAGUY (15.0 RTM) DATAGUY\DATAGUY (51)

- SQL Case functions

The SQL CASE function is a conditional expression that allows you to perform conditional logic within your SQL queries. It can be used to perform different actions based on specified conditions. There are two forms of the CASE function: simple CASE and searched CASE.

Here is an example of a simple case structure

```
SELECT column_name,
    CASE expression
        WHEN value1 THEN result1
        WHEN value2 THEN result2
        ...
        ELSE result
    END AS alias
FROM table_name;
```

```
72
73  SELECT JobTitle,
74         CASE JobTitle
75              WHEN 'Production Technician' THEN 'Entry Level'
76              WHEN 'Marketing Specialist' THEN 'Mid Level'
77              ELSE 'Unknown Level'
78         END AS JobLevel
79  FROM HumanResources.Employee;
80
81
```

100 %

Results | Messages

	JobTitle	JobLevel
1	Chief Executive Officer	Unknown Level
2	Vice President of Engineering	Unknown Level
3	Engineering Manager	Unknown Level
4	Senior Tool Designer	Unknown Level
5	Design Engineer	Unknown Level
6	Design Engineer	Unknown Level
7	Research and Development Manager	Unknown Level
8	Research and Development Engineer	Unknown Level
9	Research and Development Engineer	Unknown Level
10	Research and Development Manager	Unknown Level
11	Senior Tool Designer	Unknown Level
12	Tool Designer	Unknown Level

SQL MINI Project

Using the AdventureWorks2019 Database to answer business questions for Mzglobal.

1. We want to prepare for next year's budget and we would like to know the total sales for each year in the organization.
2. We would like to know the top 5 products that generates the most sales in quantity.
3. Retrieve employee information with department Names, job title, level in the organization, GroupName and ID of their department
4. We would like to find out the average price of our business category to enable us work on our sales strategy.
5. What is the total revenue for each sales terriory and which territory did we make the most sales?

SQL

The Future of SQL

SQL’s future is driven by AI, automation, real-time insights, and seamless integration with emerging technologies, enabling smarter, faster, and more collaborative data-driven decision-making through advanced query optimization, automation, and cloud-based solutions.

-

Magdalene