



Data Base



Hello ,I'm EN/Ahmed Hany

In This File ,I will Explain Data Base

📌 My personal accounts links

 LinkedIn	https://www.linkedin.com/in/ahmed-hany-899a9a321? utm_source=share&utm_campaign=share_via&utm_content=profile&utm_medium=android_app
 WhatsApp	https://wa.me/qr/7KNUQ7ZI3KO2N1
 Facebook	https://www.facebook.com/share/1NFM1PfSjc/

1 Basics



What is Data Base ?

A **database** is an organized collection of data that is stored and managed electronically. It allows users to efficiently store, retrieve, update, and manage data. Databases are used in various applications, from simple data storage to complex systems like banking, healthcare, and e-commerce.

A **Database Management System (DBMS)**, such as **SQL Server, MySQL, or Oracle**, is software that enables users to interact with databases.

◆ Importance of Databases:

- ✓ **Data Organization:** Stores data in a well-structured manner instead of keeping it scattered.
- ✓ **Efficient Access & Retrieval:** Enables fast data access using **SQL (Structured Query Language)**.
- ✓ **Minimizing Redundancy:** Through **Normalization**, databases reduce data duplication, improving efficiency.
- ✓ **Security & Access Control:** Allows setting **user permissions** to prevent unauthorized access.
- ✓ **Integration with Systems:** Used in **hospitals, banks, e-commerce platforms, and reservation systems** to manage large datasets.
- ✓ **Data Analysis & Decision-Making:** Helps businesses analyze data and generate reports for better decision-making.



What is Types of Data Base?

Type	Description	Examples
Relational Databases (RDBMS)	Uses tables with rows and columns, and relationships between data using primary and foreign keys .	SQL Server, MySQL, PostgreSQL, Oracle
NoSQL Databases	Stores data in flexible formats like documents, key-value pairs, wide columns, or graphs .	MongoDB (Document), Redis (Key-Value), Cassandra (Wide-Column), Neo4j (Graph)
Distributed Databases	Spreads data across multiple servers to improve scalability and availability .	Google Spanner, Amazon DynamoDB, Apache Cassandra
Object-Oriented Databases	Stores data as objects instead of tables, suitable for OOP-based applications .	ObjectDB, db4o
Time-Series Databases	Optimized for time-stamped data such as sensor logs, stock prices, and system monitoring.	Influx DB, Timescale DB
Cloud Databases	Hosted on cloud platforms, offering scalability, security, and easy access .	Amazon RDS, Google Cloud Firestore, Azure SQL Database
Embedded Databases	Small, lightweight databases that run within applications without needing a separate server.	SQLite, Berkeley DB

We Study in this file a Relational Databases



What is a Data Base Life Cycle ?

The

Database Life Cycle (DBLC) is a structured process for designing, developing, implementing, and maintaining a database system. It ensures the database meets user requirements and operates efficiently.

Step	Process	Who is Responsible?	Normalization Applied?
1. Analysis	System analysis to define requirements and data needs.	System Analyst	X No
2. Database Design (ERD)	Creating the Entity-Relationship Diagram (ERD) .	Database Designer	✓ Partial (Initial structure)
4. Database Mapping	Converting the normalized ERD into relational tables with primary & foreign keys .	Database Designer	X No (Normalization is already done)
3. Normalization	Removing redundancy, ensuring consistency, and applying Normal Forms (1NF, 2NF, 3NF, BCNF) .	Database Designer	✓ Yes, fully applied here
5. Database Implementation	Implementing the relational schema in RDBMS (SQL Server, MySQL) .	Database Developer / DBA	X No
6. Database Administration	Managing security, backup, performance tuning, and user access.	Database Administrator (DBA)	X No
7. Data Science & Analytics	Extracting, analyzing, and visualizing data for decision-making.	Data Scientist / Data Analyst	X No



What is a Problems in File base Systems ?

A **File-Based System (FBS)** stores data in separate files instead of a structured database, leading to several problems:

- 1 **✗ Data Redundancy** – The same data is stored in multiple files, wasting space.
- 2 **✗ Data Inconsistency** – Changes in one file might not be updated in others, causing mismatches.
- 3 **✗ No Data Integrity** – No rules to ensure data is valid (incorrect formats or missing values).
- 4 **✗ Limited Data Sharing** – Files are independent, making it hard to share and update data across systems.
- 5 **✗ Poor Security** – No user authentication or role-based access; anyone can access files.
- 6 **✗ Difficult Data Retrieval** – Searching for specific data is slow and requires manual processing.
- 7 **✗ No Backup & Recovery** – No automatic backup, so data loss is a big risk.

📌 Summary: Why Replace FBS with DBMS?

Problem	File-Based System (FBS)	Database (DBMS)
Data Redundancy	✓ High	✗ Low
Data Consistency	✗ Poor	✓ Strong
Data Integrity	✗ No checks	✓ Constraints
Data Sharing	✗ Limited	✓ Easy access
Security	✗ Weak	✓ Strong encryption
Multi-User Support	✗ No	✓ Yes
Searching & Queries	✗ Manual search	✓ SQL Queries
Transaction Handling	✗ No ACID properties	✓ Supports ACID



What is Difference between Data and Metadata?

Aspect	Data	Metadata
Definition	Raw facts or information stored in a system.	"Data about data" that describes the data.
Purpose	Used for processing and decision-making.	Provides structure, meaning, and context to data.
Example	A student's name, age, and grades.	Field names, data type ("Name is a text field").
Storage	Stored in databases, files, or records.	Stored as descriptions, headers, or schemas.

- ◆ **Data** → "What is stored?" (Actual content)
- ◆ **Metadata** → "How is it stored & described?" (Data about the content)

Example:

- Data:** "ahmed, 20, "Database Administrator"
- Metadata:** "Name (String), Age (Integer), Department (String)" ⇒ERD
- Metadata helps understand data**



What is Difference Between Primary Key & Foreign Key ?

Aspect	Primary Key 🔑	Foreign Key 🔗
Definition	Uniquely identifies each record in a table.	References the primary key of another table.
Uniqueness	Must be unique for each row.	Can have duplicate values.
NULL Values	Cannot be NULL.	Can be NULL.
Purpose	Ensures each record is unique.	Creates a relationship between two tables.
Example	StudentID in a Students table.	StudentID in a Courses table referencing the Students table.

Example:

◆ Students Table (Primary Key)

StudentID (PK)	Name	Age
1	omar	20
2	ali	22

◆ Courses Table (Foreign Key)

CourseID	CourseName	StudentID (FK)
101	Math	1
102	Science	2

💡 Summary:

✓ Primary Key → Ensures unique records in a table.

✓ Foreign Key → Links tables together by referencing a Primary Key.



What is Difference Between Database, DBMS, and DBS

Aspect	Database (DB)	DBMS	DBS
Definition	A structured collection of data.	Software used to manage databases.	The complete system that includes DB, DBMS, hardware, and users.
Purpose	Stores data in an organized way.	Provides tools to create, manage, and manipulate databases.	A complete environment for database management.
Contains	Only raw data (tables, records).	Tools for data manipulation, security, and transactions.	DB + DBMS + Hardware + Users.
Example	A company's customer data.	MySQL, SQL Server, Oracle DB.	A bank's entire database system, including servers and software.

Simple Explanation:

- ✓ **Database (DB)** → Just the data (like a book).
- ✓ **DBMS** → The software that manages the data (like a librarian).
- ✓ **DBS** → The entire system (library + books + librarian + users).

Analogy:

- **DB** = A collection of books.
- **DBMS** = The software that organizes and manages books.
- **DBS** = The entire library system, including books, staff, and management.



What is is ERD and How can draw it ?

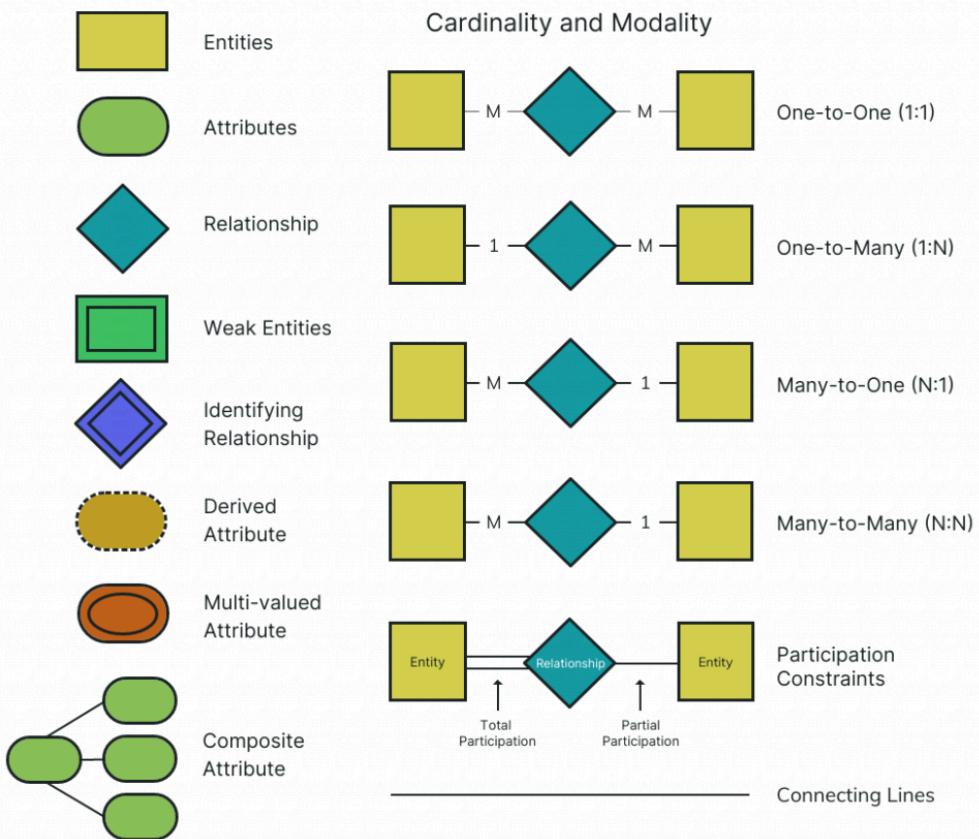
A **Entity-Relationship Diagram (ERD)** is a visual representation of a database's structure. It shows entities (tables), their attributes (columns), and the relationships between them. ERDs help in designing and understanding databases.

- **Entities:** Objects or things in a database (Patient, Doctor, Nurse).
- **Attributes:** Characteristics or properties of an entity (Name, Age, Email).
- **Relationships:** Connections between entities (A Patient visits a Doctor).
- **Attributes in Relationship** is a shared attribute between Entities

To draw an ERD from documentation provided by a data analyst, follow these simple steps:

1. **Identify Entities** – Find the main objects (nouns) in the documentation, like "Patient," "Doctor," or "Appointment."
2. **Determine Attributes** – List the key details of each entity, such as "Name," "Email," or "Date."
3. **Define Relationships** – Identify how entities relate (verbs) ("A Patient books an Appointment with a Doctor").
4. **Draw the Diagram** – Use rectangles for entities, ovals for attributes, and lines to show relationships.
5. **Refine and Review** – Ensure all necessary details are included and relationships are correct.

ERD Symbols and Notations





What is Difference Between Strong Entity and Weak Entity ?

1. Strong Entity

- An entity that can exist **independently** in the database.
- Has a **primary key** that uniquely identifies it.
- Example: Doctor (DoctorID), Patient (PatientID)

2. Weak Entity

- Cannot exist without a related **strong entity**.
- Does **not** have a **primary key** of its own.
- Has a **partial key**, which is **not unique alone** but helps distinguish records **within the weak entity**.
- Requires a **foreign key** from the strong entity for complete identification.
- The **primary key** of the weak entity is formed by combining the **partial key** with the **strong entity's primary key**.
- Example: Appointment (AppointmentNumber is a partial key, full identifier is PatientID + AppointmentNumber).

📌 **Partial Key:** A weak entity contains a **partial key** that helps distinguish records within it, but it needs the strong entity's key to be unique.

📌 **ERD Representation:** Weak entities are shown with **double rectangles**, and their relationships with strong entities are shown with **double diamonds**.



What is Types of Attributes in ERD?

1. Simple (Atomic) Attribute

- Cannot be divided further.
- **Example:** FirstName, Age, Salary.

2. Composite Attribute

- Can be divided into smaller sub-parts.
- **Example:** FullName → (FirstName, LastName).

3. Derived Attribute

- Calculated from other attributes.
- **Example:** Age (calculated from DateOfBirth).
- **ERD Representation:** Dashed oval.

4. Multivalued Attribute

- Can have multiple values for a single entity.
- **Example:** PhoneNumbers (a person can have multiple phone numbers).
- **ERD Representation:** Double oval.

5. Key Attribute

- Used to uniquely identify an entity (Primary Key).
- **Example:** PatientID, DoctorID .
- **ERD Representation:** Underlined oval.

6. Stored Attribute

- Directly stored in the database (not derived).
- **Example:** DateOfBirth .

7. Complex Attribute

- A combination of **composite** and **multivalued** attributes.
- **Example:** Address → (Street, City, State) and can have **multiple values** (home address, work address).
- **ERD Representation:** A combination of composite and multivalued representations.

📌 **ERD Representation:** Attributes are represented as **ovals** connected to entities with **lines**.



What is a Properties of Relation ?

1. Degree of Relationships
2. Cardinality Constraint
3. Participation Constraint



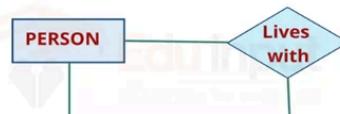
What is Degree of Relationships?

The **degree of a relationship** refers to the number of entities involved in the relationship.

1. Unary Relationship (Degree 1)

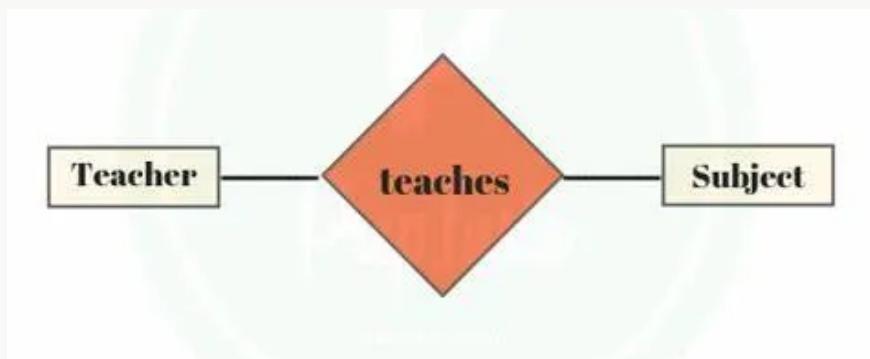
- A relationship between an entity and itself (recursive).
- **Arabic:** يسْعَانْ تَعْمَلْ كَدَا لَازِمْ Two Entities need to exist for a relationship to occur.
- **Example:** Employee manages Employee .

Unary Relationship



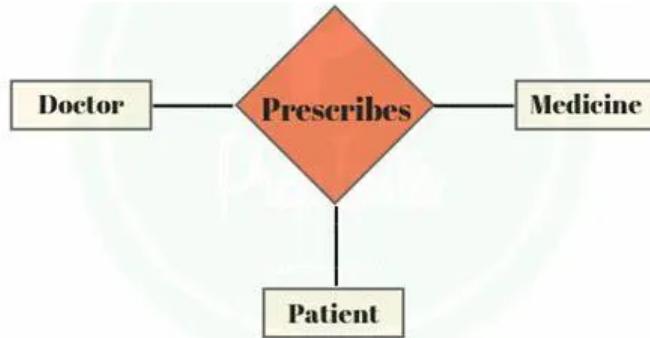
2. Binary Relationship (Degree 2)

- A relationship between **two entities**.
- **Example:** Doctor treats Patient .



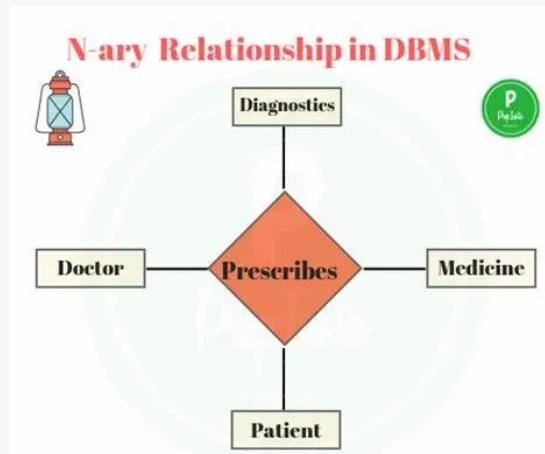
3. Ternary Relationship (Degree 3)

- A relationship between **three entities**.
- **Example:** Doctor , Patient , and Medicine (a doctor prescribes medicine to a patient).



4. N-ary Relationship (Degree N)

- A relationship between **more than three entities**.
- **Example:** Student, Course, Instructor, and Classroom (a student enrolls in a course taught by an instructor in a specific classroom).



📌 **ERD Representation:** Relationships are represented by **diamonds** connecting entities.



What is Types of Cardinality Constraints ?

A **Cardinality Constraint** defines the **number of instances** an entity can be associated with another entity in a relationship.

Types of Cardinality Constraints

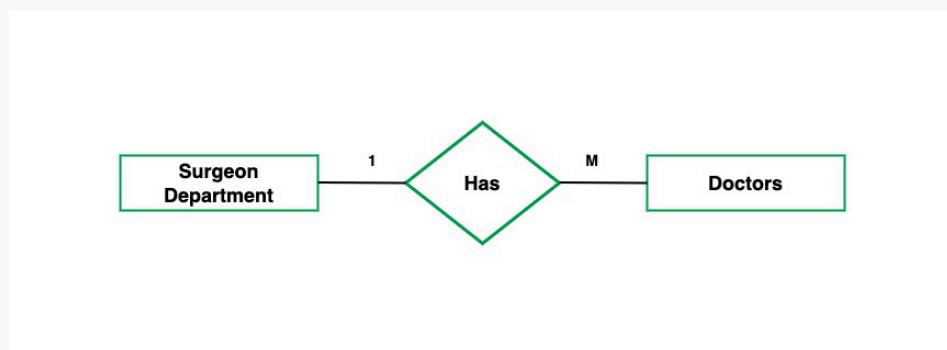
1. One-to-One (1:1)

- Each entity in Set A is related to **at most one** entity in Set B, and vice versa.
- **Example:** A **passport** is assigned to only **one person**, and each person has only **one passport**.



2. One-to-Many (1:M)

- Each entity in Set A is related to **multiple** entities in Set B, but each entity in Set B is related to **only one** entity in Set A.
- **Example:** A **doctor** can treat **multiple patients**, but each **patient** has only **one doctor**.



3. Many-to-Many (M:N)

- Each entity in Set A can be related to **multiple** entities in Set B, and vice versa.
- **Example:** **Students** enroll in **multiple courses**, and each **course** has multiple **students**.



📌 **ERD Representation:**

- Cardinality is represented as **(1, M, N)** next to the relationship lines.



What is a Types of Participation Constraint in ERD?

A **Participation Constraint** specifies whether all entities in an entity set must participate in a relationship. It can be **Total** or **Partial**.

1. Total Participation

- Every instance of the entity **must** participate in the relationship.
- **Example:** Every **patient** **must** be assigned to a **doctor**.
- **any weak entity must be Total Participation**
- **ERD Representation:** Double line connecting the entity to the relationship.

2. Partial Participation

- Some instances of the entity **may** participate in the relationship, but not all.
- **Example:** Not every **doctor** is assigned to a **patient** (some may be researchers).
- **ERD Representation:** Single line connecting the entity to the relationship.



- Student must Enrolled in at least one course
- course can optional contain students

Total vs. Partial Participation (Simple Words)

◆ **Total Participation** → **Must** be in the relationship

Words: "Every," "All," "Must," "Required", "Mandatory"

📌 **Example:** "Every student must enroll in a course."

◆ **Partial Participation** → **May or may not** be in the relationship

Words: "Some," "May," "Can," "Optional"

📌 **Example:** "Some doctors may not treat any patients."



What is Different Types of Keys in Database ?

1. Primary Key (PK)

- Uniquely identifies each record in a table.
- Example: `StudentID` in a **Students** table.

2. Unique Key

- Ensures a column has unique values but allows **one NULL** value only.
- Example: `Email` in a **Users** table.

Example:

UserID (PK)	Email (Unique Key)
1	user1@email.com
2	user2@email.com
3	NULL (Allowed)
4	user4@email.com

Difference:

- Primary Key (PK) **NULL not allowed**
- Unique Key **One NULL allowed**

3. Super Key

- A set of attributes that uniquely identify a record (can have extra attributes).
- Example: `StudentID + Name` (still unique, but Name is extra).

4. Foreign Key (FK)

- A key that refers to a primary key in another table **can be null**
- can not delete row that has a foreign key at another table by default
- Example: `DepartmentID` in **Employee** (links to **Department** table).

5. Composite Key

- A primary key made of **two or more** attributes.
- Example: `StudentID + CourseID` in **Enrollment** table.

6. Candidate Key

- A set of attributes that can be a primary key.
- Example: `Email` and `StudentID` (either could be PK).

7. Alternate Key

- A candidate key **not chosen** as the primary key.
- **Example:** If `StudentID` is PK, then `Email` is an alternate key.

8. Partial Key (Discriminator Key) (For Weak Entities)

- Helps identify a weak entity but needs a strong entity's key.
- **Example:** `AppointmentNumber` in **Appointments** (needs `PatientID`).

الخلاصة:

- **Primary Key** → Main unique key.
- **Candidate Key** → Multiple options for PK.
- **Super Key** → Unique but may have extra columns.
- **Foreign Key** → Links to another table.
- **Composite Key** → Made of multiple attributes.
- **Alternate Key** → A candidate key not used as PK.
- **Unique Key** → Ensures uniqueness (allows NULL).
- **Partial Key** → Identifies weak entities (with FK).

Mapping



What is a steps of Mapping?

1. Mapping of Regular Entity Types
2. Mapping of Weak Entity Types
3. Mapping of Binary 1:1 Relation Types
4. Mapping of Binary 1:N Relationship Types.
5. Mapping of Binary M:N Relationship Types.
6. Mapping of N-ary Relationship Types.
7. Mapping of Unary Relationship.



How to Mapping of Regular Entity & Attributes?

1 Mapping of Regular Entity

- Create a **separate table** for each strong entity.
- The **Primary Key (PK)** uniquely identifies each record.
- All **attributes** become **columns** in the table.

✓ Example:

- Doctor(DoctorID, Name, Specialization, ExperienceYears)

2 Mapping of Attributes

📌 Simple Attributes:

- Stored **directly** as columns.
- **Example:** Name, Specialization, ExperienceYears

📌 Composite Attributes:

- Split into **multiple columns**.
- **Example:** Patient(Name → FirstName, LastName)

📌 Multivalued Attributes:

- Stored in a **separate table**.
- **Example:** Doctor(PhoneNumbers can have multiple values)

📌 Derived Attributes:

- **Not stored**, computed when needed.
- **Example:** Age (calculated from DateOfBirth)

📌 Complex Attributes:

- **Mix** of composite & multivalued attributes.
- **Example:** Doctor(Address → (Street, City, State, ZipCode) + Multiple Clinics)

Example:

📌 Hospital Management System

We will define **tables** based on the different **types of attributes** in the **Doctor** entity.

1 Doctor (Regular Entity)

- DoctorID (**Primary Key**)

- **Name** (*Simple*)
- **Specialization** (*Simple*)
- **ExperienceYears** (*Simple*)
- **DateOfBirth** (*Simple*)
- **Age** (*Derived, not stored in the table*) it calculated at runtime

2 Address (Composite Attribute – Stored Separately)

Since `Address` is composite, it is stored in separate columns:

- **DoctorID** (*Foreign Key → References Doctor*)
- **Street**
- **City**
- **State**
- **ZipCode**

3 DoctorPhoneNumbers (Multivalued Attribute – Separate Table)

Since a doctor can have **multiple phone numbers**, we store them in a separate table:

- **DoctorID** (*Foreign Key → References Doctor*)
- **PhoneNumber** (*Each doctor may have multiple numbers*)

4 WorkLocations (Complex Attribute – Composite + Multivalued)

A doctor may work in **multiple hospitals**, each with its own address.

- **DoctorID** (*Foreign Key → References Doctor*)
- **HospitalName** (*Name of the hospital*) ⇒ **Multivalued**
- **Street**
- **City**
- **State**
- **ZipCode**

Final Tables Structure

Table Name	Columns
Doctor	DoctorID (PK), Name, Specialization, ExperienceYears, DateOfBirth
Address	DoctorID (FK), Street, City, State, ZipCode
DoctorPhoneNumbers	DoctorID (FK), PhoneNumber



How to Mapping of Weak Entity Types?

- Weak Entity depends on a Strong Entity and cannot exist alone.
- It has a Partial Key + Foreign Key from the strong entity.
- Primary Key = (Foreign Key of Strong Entity + Partial Key).

Example

Entities:

- 1 Patient (Strong Entity) → PatientID (PK)
- 2 Appointment (Weak Entity) → AppointmentNo (Partial Key)

Tables

1 Patient (Strong Entity)

PatientID (PK)	Name
101	Ali
102	Sara

2 Appointment (Weak Entity)

PatientID (FK, PK)	AppointmentNo (PK, Partial Key)	Date
101	1	2024-02-10
101	2	2024-02-15
102	1	2024-02-12



How can Mapping of Binary 1:1 Relation Types?

1. Case 1: Both Sides Mandatory (Total Participation)
 - Merge both entities into a single table.
 - Use one primary key (PK) that represents both.
 - Include attributes from both entities.
2. Case 2: One Side Mandatory, One Optional
 - Create a table for Entity that is partial with another entity
 - create a table for Entity that is mandatory of another entity and contain a FK from Entity that is in the side of optional



here we create a table of course with PK

and create a table of student that contain his PK and FK from Course

3. Case 3: Both Sides Optional (Partial Participation)
 - Create two separate tables, one for each entity.
 - create FK in one table and allow null in FK or :
 - create the table three that contain a FK from every table and one of its will be a PK not two keys only one key of two keys

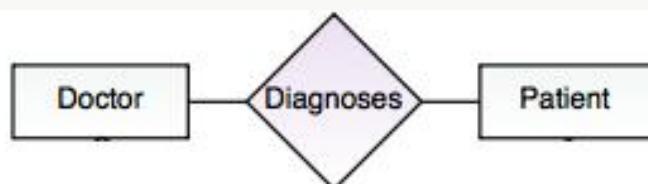


Fig. Partial Participation

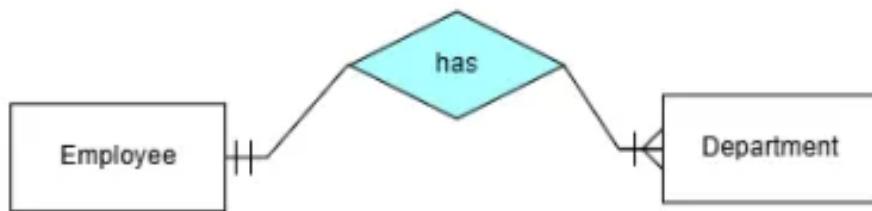
here we create table for Doctor and Table for patient and Doctor_Patient table that contain
PK & FK⇒doctor-id and FK ⇒ Patient-id
or put in patient table a fk of doctor that allow null



How can Mapping of Binary 1:N Relationship Types?

we don't care about one we only look at many side if it :

1. Many Mandatory \Rightarrow create a table of the Entity in one side and Create a table of Entity that in Many side and will contain a FK from the another table of one side

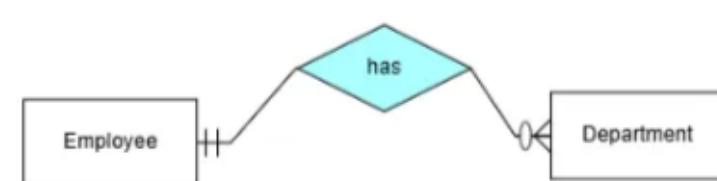


here we create a table for Employee contain PK of it

and Create a Table of Department that contain PK of Department and FK of Employee

here FK can be null

1. Many is Optional \Rightarrow create a table of one side and table of many side and table of together that contain a PK ,FK from many side and FK from one side
2. here FK can not be null

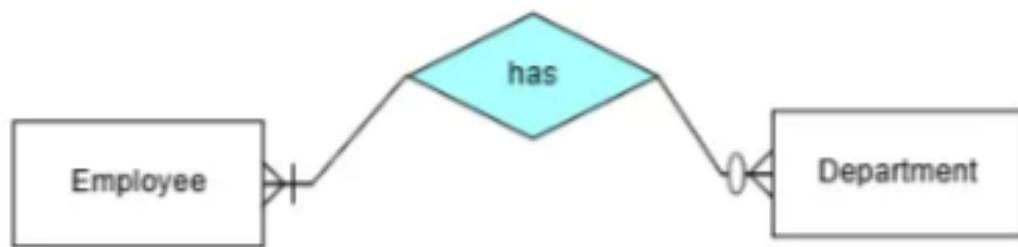


create table for employee and table for department and table emp_depart contain
PK ,FK \Rightarrow departmentId and FK \Rightarrow employeeid



How can Mapping of Binary M:N Relationship Types?

don't look to participation start create a table for every entity and a table that contain a composite PK of two FKs from two tables



create a table for employee \Rightarrow pk(empid)

create a table for department \Rightarrow pk(departid)

create a table for emp_depart \Rightarrow pk(empid,departid) \Rightarrow composite key



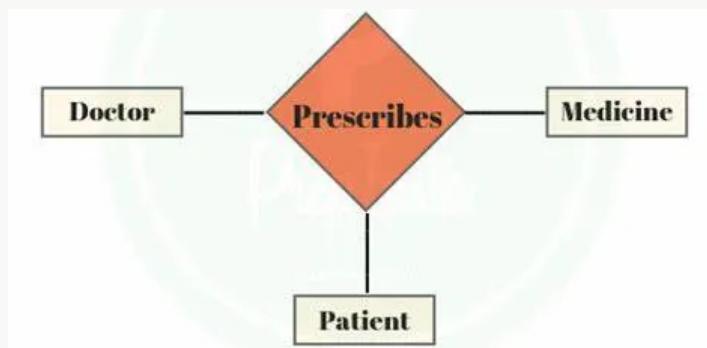
How can Mapping of N-ary Relationship Types?

we don't look at cardinality or participation , we create 4 tables

one table for every entity and table contain FK from every table \Rightarrow 3 FKs

and choose a PK from your FKs or by merge FK with a Attributes (super key) that in a Relation

or if no one correct create a new column to be a PK



here we create a table for Doctor and table for Medicine and table for Patient
and table contain FKs(Doctorid,patientid,medicineid) and new column to be PK



How to Mapping Unary Relationship ?

create one table contain a PK of table and FK of Unary Relation (Recursive)

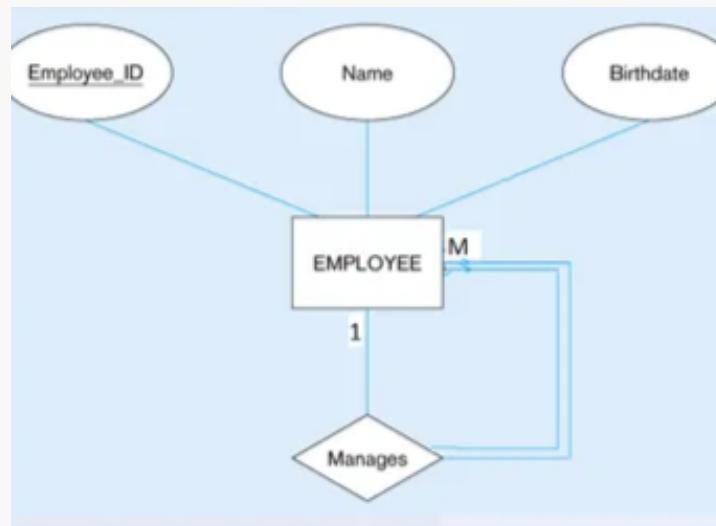
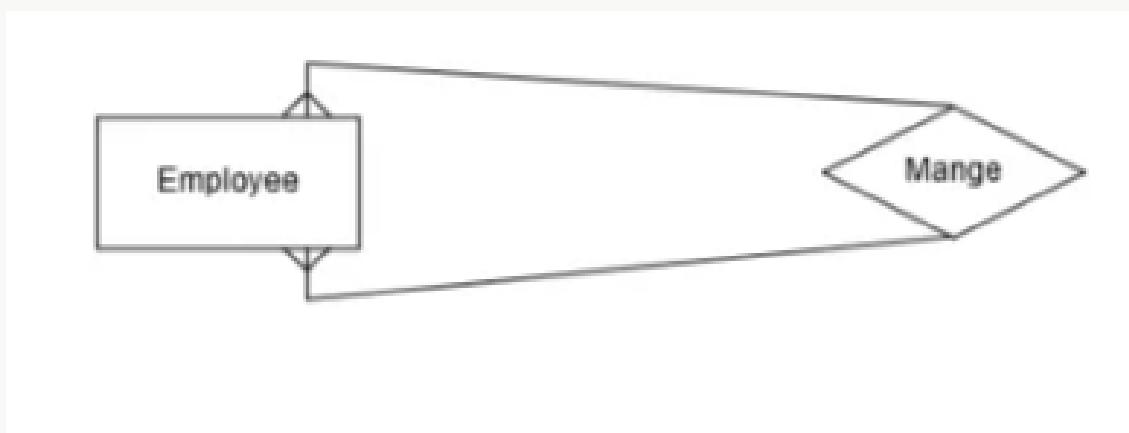


table contain [PK(Employee_ID), Name , Birthdate , FK(Manager_ID)]

Fk(Manager_ID) pointer to ⇒ PK(Employee_ID)

If you have self Relation many to many what can you do ?

create basic table and create another table contain Two Fks and create new column to be PK



create table Employee contain PK(Employeeid) and create Table contain Employeeid and Managerid and id ⇒ to be PK

1. جدول الموظفين (Employee)

يحتوي على جميع الموظفين والمدربين.

Id	Name	Position
1	أحمد	مدير عام
2	محمد	مدير قسم
3	سارة	موظف
4	علي	موظف
5	خالد	موظف

2. (جدول العلاقات EmployeeManager)

يربط كل موظف بمعديره المباشر (جدول وسيط)

Id	EmployeeId	ManagerId
1	2	1
2	3	2
3	4	2
4	5	1

SQL



What is ANSI SQL?

ANSI SQL is the **standard version of SQL** that defines basic rules for working with relational databases. It ensures that SQL commands like `SELECT`, `INSERT`, `UPDATE`, and `DELETE` work the same way across different database systems (like MySQL, Microsoft Transact SQL Server, PostgreSQL, and Oracle).

Why is ANSI SQL important?

- ✓ **Portability:** You can use the same SQL queries in different databases.
- ✓ **Consistency:** It provides a common way to write and understand SQL.
- ✓ **Compatibility:** Makes it easier to switch between database systems.

Example of ANSI SQL Query (Works Everywhere)

```
SELECT FirstName, LastName  
FROM Employees  
WHERE Department = 'IT';
```

 In short, ANSI SQL is the "official" SQL language standard!



What is Microsoft Transact SQL Server Categories ?

In **Microsoft Transact-SQL (T-SQL)**, the SQL commands are categorized into five main types:

1 DDL (Data Definition Language) – Defines Structure

Used to create and modify database schema.

- `CREATE TABLE` – Creates a new table
- `ALTER TABLE` – Modifies an existing table
- `DROP TABLE` – Deletes a table
- `CREATE INDEX` – Creates an index
- `DROP INDEX` – Deletes an index

2 DML (Data Manipulation Language) – Modifies Data

Used to insert, update, delete, and manipulate records.

- `INSERT` – Adds new records
- `UPDATE` – Modifies existing records
- `DELETE` – Removes records

3 DQL (Data Query Language) – Retrieves Data

Used to fetch data from the database.

- `SELECT` – Retrieves data from tables and views
- `SELECT DISTINCT` – Returns unique values
- `WHERE` – Filters records
- `ORDER BY` – Sorts records
- `GROUP BY` – Groups data

4 DCL (Data Control Language) – Manages Permissions

Controls access to data.

- `GRANT` – Gives permissions to users
- `REVOKE` – Removes permissions

5 TCL (Transaction Control Language) – Manages Transactions

Used to control transactions in SQL Server.

- `BEGIN TRANSACTION` – Starts a transaction
- `COMMIT` – Saves changes
- `ROLLBACK` – Reverts changes
- `SAVE TRANSACTION` – Creates a savepoint in a transaction

Summary:

- **DDL** – Defines tables (`CREATE` , `ALTER` , `DROP`)
- **DML** – Modifies data (`INSERT` , `UPDATE` , `DELETE`)
- **DQL** – Queries data (`SELECT` , `WHERE` , `GROUP BY`)
- **DCL** – Controls access (`GRANT` , `REVOKE`)
- **TCL** – Manages transactions (`COMMIT` , `ROLLBACK`)



Data Types :

1 Numeric Data Types (الأنواع العددية)

Data Type	Storage (Bytes)	Range
BIT	1	0 or 1 (True/False)
TINYINT	1	0 to 255
SMALLINT	2	-32,768 to 32,767
INT	4	-2,147,483,648 to 2,147,483,647
BIGINT	8	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
DECIMAL(p, s) / NUMERIC(p, s)	5-17	Fixed precision & scale
FLOAT(n)	4 or 8	Approximate floating-point
REAL	4	Approximate floating-point

2 Character (String) Data Types (الأنواع النصية)

Data Type	Storage	Description
CHAR(n)	Fixed (1 to 8,000)	Fixed-length string
VARCHAR(n)	Variable (1 to 8,000)	Variable-length string
TEXT	Variable (deprecated)	Large text data
NCHAR(n)	Fixed (1 to 4,000)	Fixed-length Unicode string
NVARCHAR(n)	Variable (1 to 4,000)	Variable-length Unicode string
NTEXT	Variable (deprecated)	Large Unicode text

3 Date & Time Data Types (الأنواع الزمنية)

Data Type	Storage (Bytes)	Description
DATE	3	Stores only date (YYYY-MM-DD)
TIME(p)	3-5	Stores only time (HH:MI:SS)
DATETIME	8	Stores both date & time
DATETIME2(p)	6-8	More precise DATETIME
SMALLDATETIME	4	Stores date & time (less precision)

DATETIMEOFFSET	10	Stores date & time with time zone
----------------	----	-----------------------------------

4 Binary Data Types - الصور والملفات (البيانات الثنائية)

Data Type	Storage	Description
BINARY(n)	Fixed (1 to 8,000)	Fixed-length binary data
VARBINARY(n)	Variable (1 to 8,000)	Variable-length binary data
IMAGE	Variable (deprecated)	Stores large binary objects

5 Special Data Types - أنواع خاصة

Data Type	Storage	Description
SQL_VARIANT	Up to 8,000	Stores multiple data types in one column
UNIQUEIDENTIFIER	16	Globally Unique Identifier (GUID)
XML	Variable	Stores XML data
CURSOR	Variable	Stores cursor reference
TABLE	Variable	Stores a table inside a variable

6 Deprecated Data Types - أنواع غير مفضلة

♦ لم يعد يُنصح باستخدام هذه الأنواع، وقد يتم إزالتها في الإصدارات المستقبلية من SQL Server:

- TEXT (استبدل بـ VARCHAR(MAX))
- NTEXT (استبدل بـ NVARCHAR(MAX))
- IMAGE (استبدل بـ VARBINARY(MAX))

♦ Summary Table:

Category	Examples
Numeric	INT, BIGINT, FLOAT, DECIMAL
Character	CHAR, VARCHAR, TEXT, NCHAR, NVARCHAR
Date/Time	DATE, DATETIME, TIME, DATETIMEOFFSET
Binary	BINARY, VARBINARY, IMAGE
Special	XML, CURSOR, TABLE, SQL_VARIANT

💡 اختيار نوع البيانات المناسب يحسن الأداء ويوفر المساحة 🚀



1 DDL

```
--DDL
--create
CREATE DATABASE FDB;--for create new Data base
USE FDB;--for choose your database you will work in it

CREATE TABLE Employee
(
id int primary key,
[Name] varchar(50) default 'employee',
--here Name has a default value
age int not null, --age can not be null,
Birthdata date,
);
--alter
alter table Employee add salary int ;
alter table Employee alter column salary float;

alter table Employee drop column salary ; --drop column

--Delete
drop table Employee; --deleting table
drop DATABASE FDB; --deleting database
```



DML

--DML

--insert one row

```
INSERT INTO Employee  
VALUES (1, 'ahmed', 20, '2004-07-27');
```

```
INSERT INTO Employee (id,age,[Name],Birthdata) VALUES  
(2,25,'ali','2000-05-24');
```

-- insert constructot

```
INSERT INTO Employee VALUES
```

```
(3,'omar',21,'2003-04-12'),  
(4,'yasser',21,'2003-04-12')
```

--update in table

```
update Employee
```

```
set [Name] = 'ashraf'
```

```
where id = 3
```

--delete row

```
DELETE From Employee where id=4;
```



DQL

--DQL

SELECT * FROM Employee; --Display all

--projection

SELECT id , [Name] FROM Employee;

SELECT * FROM Employee where age >20

--ORDER Columns by default asending

SELECT * FROM Employee order by age; --here ids not order

--ORDER desinding

SELECT * FROM Employee order by age DESC;

--not repeated data

SELECT distinct age FROM Employee

--and , or , in ,between

SELECT * FROM Employee where [Name]='ahmed' and id =1;

SELECT * FROM Employee where [Name]='ahmed' or id =1;

SELECT * FROM Employee where age in (21,22) --same of or bu deference values

SELECT * FROM Employee where age between 20 and 24;



Types of Join

نوع JOIN	الصفوف المسترجعة
INNER JOIN	فقط الصفوف المتطابقة بين الجدولين
LEFT JOIN	كل الصفوف من الجدول الأيسر + المطابقات من الجدول الأيمن
RIGHT JOIN	كل الصفوف من الجدول الأيمن + المطابقات من الجدول الأيسر
FULL JOIN	كل الصفوف من كلا الجدولين (NULL بالمخابقات والـ)
CROSS JOIN	الجداء الديكارتي (كل صف في الأول × كل صف في الثاني)
SELF JOIN	يستخدم للربط بين بيانات الجدول نفسه

Testing

-- إنشاء قاعدة البيانات

```
CREATE DATABASE CompanyDB;
USE CompanyDB;
```

-- إنشاء جدول الأقسام

```
CREATE TABLE Department (
    DepartmentID INT PRIMARY KEY,
    DepartmentName VARCHAR(50)
);
```

-- إدخال بيانات الأقسام

```
INSERT INTO Department (DepartmentID, DepartmentName) VALUES
(1, 'HR'),
(2, 'IT'),
(3, 'Finance'),
(4, 'Marketing');
```

-- إنشاء جدول الموظفين

```
CREATE TABLE Employee (
    EmployeeID INT PRIMARY KEY,
    Name VARCHAR(50),
    Age INT,
```

```

يمكن أن يكون بعض الموظفين بدون قسم -- 
ManagerID INT NULL, -- لدعم SELF JOIN
FOREIGN KEY (DepartmentID) REFERENCES Department(DepartmentID),
FOREIGN KEY (ManagerID) REFERENCES Employee(EmployeeID)
);

إدخال بيانات الموظفين --
INSERT INTO Employee (EmployeeID, Name, Age, DepartmentID, ManagerID) VALUES
(1, 'Ahmed', 30, 1, NULL), -- بدون مدير HR موظف في
(2, 'Sara', 25, 2, 1), -- ومديريها أحمد IT موظفة في
(3, 'Ali', 35, 3, 1), -- ومديره محمد موظف في
(4, 'Mona', 28, 2, 3), -- ومديريها علي IT موظفة في
(5, 'Omar', 40, NULL, NULL), -- موظف بدون قسم وبدون مدير --
(6, 'Laila', 29, 4, NULL), -- بدون مدير Marketing موظفة في
(7, 'Khaled', 38, 3, 2); -- ومديريه سارة موظف Finance في

SELECT * FROM Department;
SELECT * FROM Employee;

```

1. Inner Join

```

SELECT [Name] , DepartmentName from
Employee E ,Department D
Where E.DepartmentID=D.DepartmentID

```

```

SELECT [Name] , DepartmentName from
Employee E inner join Department D
on E.DepartmentID=D.DepartmentID

```

Output

Ahmed	HR
Sara	IT
Ali	Finance
Mona	IT
Laila	Marketing
Khaled	Finance

2. Left Outer Join

```
SELECT [Name] , DepartmentName from  
Employee E left outer join Department D  
on E.DepartmentID=D.DepartmentID
```

Output

Ahmed	HR
Sara	IT
Ali	Finance
Mona	IT
Omar	NULL
Laila	Marketing
Khaled	Finance

3. Right Outer Join

```
INSERT INTO Department (DepartmentID) values (5);  
SELECT [Name] , DepartmentName from  
Employee E right outer join Department D  
on E.DepartmentID=D.DepartmentID
```

Output

Ahmed	HR
Sara	IT
Mona	IT
Ali	Finance
Khaled	Finance
Laila	Marketing
NULL	NULL

4. Full Outer Join

```
SELECT [Name] , DepartmentName from  
Employee E full outer join Department D  
on E.DepartmentID=D.DepartmentID
```

Output

Ahmed	HR
Sara	IT
Ali	Finance
Mona	IT
Omar	NULL
Laila	Marketing
Khaled	Finance
NULL	NULL

5. Cross Join

```
SELECT [Name],DepartmentName from  
Employee ,Department ;
```

```
SELECT [Name],DepartmentName from  
Employee Cross Join Department ;
```

Output

Ahmed	HR
Sara	HR
Ali	HR
Mona	HR
Omar	HR
Laila	HR
Khaled	HR
Ahmed	IT
Sara	IT
Ali	IT
Mona	IT
Omar	IT
Laila	IT
Khaled	IT
Ahmed	Finance

Sara	Finance
Ali	Finance
Mona	Finance
Omar	Finance
Laila	Finance
Khaled	Finance
Ahmed	Marketing
Sara	Marketing
Ali	Marketing
Mona	Marketing
Omar	Marketing
Laila	Marketing
Khaled	Marketing
Ahmed	NULL
Sara	NULL
Ali	NULL
Mona	NULL
Omar	NULL
Laila	NULL
Khaled	NULL

6. Self Join

```
SELECT E1.Name ,E2.Name  
FROM Employee E1,Employee E2  
Where E2.EmployeeID =E1.ManagerID  
--her E2 is a child that is a new table  
--E1 is a base table in DB that contain FK
```

Output

Sara	Ahmed
Ali	Ahmed
Mona	Ali
Khaled	Sara

use Join With Update and Delete ?

Join with Update

```
UPDATE E  
Set Age+=1  
From Employee E , Department D  
Where E.DepartmentID =D.DepartmentID
```

Output

1	Ahmed	31	1	NULL
2	Sara	26	2	1
3	Ali	36	3	1
4	Mona	29	2	3
5	Omar	40	NULL	NULL
6	Laila	30	4	NULL
7	Khaled	39	3	2

Join With Delete

```
Delete E  
From Employee E , Department D  
Where E.DepartmentID !=D.DepartmentID
```

5	Omar	40	NULL	NULL
---	------	----	------	------



What Difference between `Isnull` and `Coalesce` ?

- Use `ISNULL` for a simple **NULL replacement**.

```
SELECT isnull(ManagerID,555)
from Employee
```

output
555
1
1
3
555
555
2

- Use `COALESCE` when you need to **check multiple values** and return the first non-NULL one.

```
SELECT Coalesce(ManagerID,DepartmentID,111)
FROM Employee
```

output
1
1
1
3
111
4
2



How can Concatenation Diferent Data ?

```
Select Concat(ManagerID,DepartmentID,Name)
FROM Employee
```

output

1Ahmed

12Sara

13Ali

32Mona

Omar

4Laila

23Khaled



Like

Pattern	Description	Example Match	Not Match
a%m	"a" يبدأ بـ "m" وينتهي بـ "m"	adam , aolm	am , alex
%g	ـ "g" ينتهي بـ "g"	king , learning	go , good
%v_	ـ "v" الدرف قبل الأخير هو	live , dive	give , love
[atd]%	"a" أو "t" أو "d" يبدأ بحرف	apple , tree , dog	ball , sun
[^atd]%	"a" أو "t" أو "d" لا يبدأ بحرف	sun , moon	apple , tree
[a-r]%	"a" يبدأ بحرف بين "r"	apple , road , mark	sun , zebra
%[%]	ـ "%" ينتهي بـ "%"	50% , discount%	rate , value

◆ مع هذه الأنماط في استعلام **LIKE** استخدام:

```
SELECT * FROM TableName WHERE ColumnName LIKE 'a%m';
```

💡 هذا سيلoad كل القيم التي تبدأ بـ "a" وينتهي بـ "m".

Normalization



ودي بتجالها لما يبقى عندك الداتا بيز بالفعل بس فيها بعض المشاكل وعاوز تعدل عليها



What is Normalization?

Normalization is the process of organizing data in a database to reduce redundancy and improve data integrity by dividing large tables into smaller, related tables and defining relationships between them.



When we should apply Normalization?

Insertion Anomaly – adding new rows forces user to create duplicate data

Deletion Anomaly – deleting rows may cause a loss of data that would be needed for other future rows

Modification Anomaly – changing data in a row forces changes to other rows because of duplication



What is a Functional Dependency (FD)?

A **Functional Dependency (FD)** is a relationship between **attributes (columns)** in a database table, where the value of one attribute **determines** the value of another attribute.

💡 Example:

In a **Students** table:

- **StudentID → StudentName** (StudentID uniquely determines StudentName)

This means if you know the **StudentID**, you can always find the correct **StudentName**.



What are Types of functional dependency?

1. Full Functional Dependency

A functional dependency (FD) $X \rightarrow Y$ is **fully functional** if Y is dependent on the entire key X , not just a part of it.

Occurs in: 2NF (Second Normal Form)

Example:

In a **Student_Course** table:

$(\text{StudentID}, \text{CourseID}) \rightarrow \text{Grade}$

Here, Grade depends on **both** StudentID and CourseID together.

- If Grade depends only on StudentID , it is **not fully functional**.

2. Partial Functional Dependency

A functional dependency $X \rightarrow Y$ is **partial** if **a part of the key X alone determines Y**, rather than the full key.

Occurs in: 1NF and removed in 2NF

Example:

$(\text{StudentID}, \text{CourseID}) \rightarrow \text{Grade}$

$\text{StudentID} \rightarrow \text{StudentName}$

Here, StudentName depends only on StudentID , not CourseID .

- Since StudentID is part of the **composite key**, this is a **partial dependency**.

Fix: Move StudentName to a separate table and achieve **2NF**.

3. Transitive Functional Dependency

A **transitive dependency** happens when:

1. $X \rightarrow Y$
2. $Y \rightarrow Z$
3. Therefore, $X \rightarrow Z$ (indirect dependency)



Occurs in: 2NF and removed in 3NF

4. X, Y, Z is None Keys

◆ Example:

$\text{StudentID} \rightarrow \text{DepartmentID}$
 $\text{DepartmentID} \rightarrow \text{DepartmentName}$
So, $\text{StudentID} \rightarrow \text{DepartmentName}$ (Transitive Dependency)

📌 Fix: Move [DepartmentName](#) to a separate **Department table** and achieve **3NF**.



What is a Three Normal Forms ?

1. First Normal Form (1NF) - Remove Repeating Groups

Rule:

- ✓ Each column contains **atomic (indivisible) values**.
- ✓ Each column has a **unique name**.
- ✓ Each row must have a **unique identifier (Primary Key)**.

◆ **Example (Before 1NF - Repeating Values):**

StudentID	Name	Courses
101	John	Math, Science
102	Sara	English

◆ **Fix (After 1NF - No Repeating Values):**

StudentID	Name	Course
101	John	Math
101	John	Science
102	Sara	English

📌 **Problem in 1NF:** Redundancy still exists! (John appears twice)

2. Second Normal Form (2NF) - Remove Partial Dependency

Rule:

- ✓ Must be in **1NF**.
- ✓ **No partial dependencies** (All **non-key attributes** should depend on the **whole primary key**, not just a part of it).

◆ **Example (Before 2NF - Partial Dependency):**

StudentID	CourseID	StudentName	Grade
101	MATH101	John	A
101	SCI102	John	B

📌 **Issue:** `StudentName` depends only on `StudentID`, not `CourseID`.

◆ **Fix (After 2NF - Remove Partial Dependency):**

Students Table

StudentID	StudentName
101	John

Courses Table

CourseID	StudentID	Grade
MATH101	101	A
SCI102	101	B

 **Problem in 2NF: Still has transitive dependency!** (Department Name depends on Department ID)

3. Third Normal Form (3NF) - Remove Transitive Dependency

Rule:

- ✓ Must be in 2NF.
- ✓ **No transitive dependencies** (A **non-key attribute** should NOT depend on another **non-key attribute**).

◆ **Example (Before 3NF - Transitive Dependency):**

StudentID	StudentName	DepartmentID	DepartmentName
101	John	D1	Computer Science
102	Sara	D2	Mathematics

 **Issue:** `DepartmentName` depends on `DepartmentID`, not directly on `StudentID`.

◆ **Fix (After 3NF - Remove Transitive Dependency):**

Students Table

StudentID	StudentName	DepartmentID
101	John	D1
102	Sara	D2

Departments Table

DepartmentID	DepartmentName
D1	Computer Science
D2	Mathematics



Aggregate Functions

Function	Description	Example
COUNT()	Counts the number of rows.	SELECT COUNT(*) FROM Patients;
SUM()	Calculates the total sum of a numeric column.	SELECT SUM(Price) FROM Medications;
AVG()	Calculates the average value of a numeric column.	SELECT AVG(Salary) FROM Doctors;
MIN()	Returns the smallest value in a column.	SELECT MIN(Age) FROM Patients;
MAX()	Returns the largest value in a column.	SELECT MAX(Revenue) FROM Hospital;
STRING_AGG() (SQL Server, PostgreSQL)	Concatenates values into a single string with a specified separator.	SELECT STRING_AGG(Name, ', ') FROM Patients;

Usage with GROUP BY

To apply aggregate functions to groups of data:

```
SELECT Department, COUNT(*) AS PatientCount
FROM Patients
GROUP BY Department;
```

Usage with HAVING

To filter grouped results:

```
SELECT Department, COUNT(*) AS PatientCount
FROM Patients
GROUP BY Department
HAVING COUNT(*) > 10;
```

- WHERE تُستخدم لتصفية البيانات قبل GROUP BY .
rows بتعمل تصفية لل
- HAVING تُستخدم لتصفية البيانات بعد GROUP BY .
columns بتعمل تصفية لل

Example:

```

-- Create the database
CREATE DATABASE HospitalDB;
USE HospitalDB;

-- Create Patients Table
CREATE TABLE Patients (
    PatientID INT PRIMARY KEY identity,
    Name VARCHAR(100),
    Age INT,
    Department VARCHAR(50),
    BillAmount DECIMAL(10,2)
);

-- Insert sample data into Patients table
INSERT INTO Patients (Name, Age, Department, BillAmount) VALUES
('John Doe', 45, 'Ophthalmology', 200.50),
('Jane Smith', 30, 'Pediatrics', 150.75),
('Alice Brown', 25, 'Ophthalmology', 300.00),
('Bob Johnson', 40, 'Pediatrics', 120.00),
('Charlie White', 50, 'Ophthalmology', 400.25);

-- Create Doctors Table
CREATE TABLE Doctors (
    DoctorID INT PRIMARY KEY identity,
    Name VARCHAR(100),
    Specialization VARCHAR(50),
    Salary DECIMAL(10,2)
);

-- Insert sample data into Doctors table
INSERT INTO Doctors (Name, Specialization, Salary) VALUES
('Dr. Adams', 'Ophthalmology', 5000.00),
('Dr. Baker', 'Pediatrics', 6000.00),
('Dr. Clark', 'Ophthalmology', 5500.00),
('Dr. Davis', 'Pediatrics', 6200.00);

SELECT * From Doctors;
SELECT * From Patients;

--Count
SELECT count(*) as Total_Patients
From Patients;

```

```
--Sum
SELECT SUM(BillAmount) as Total_BillAmount
From Patients;
--Average
SELECT AVG(Age) as average_ages
FROM Patients;
--Max
SELECT MAX(BillAmount) as MaxBillAmount
from Patients;
--MIN
SELECT Min(Age) as MaxBillAmount
from Patients;
--STRING_AGG
SELECT STRING_AGG(Name, '/') AS AllDoctors FROM Doctors;

--Group By
SELECT Department, COUNT(*) as departmentpatient FROM Patients
Group by Department

SELECT Department, avg(BillAmount)as AVR_Amount FROM Patients
Group by Department

SELECT Specialization ,Max(Salary) as max_salary from Doctors
Group BY Specialization;

--using Having
SELECT Department, avg(BillAmount)as AVR_Amount FROM Patients
Group by Department
Having avg(BillAmount) > 200;
```



Sub Queries

```
USE HospitalDB;
```

--Sub Quieries

--Retrieve the names of patients who have a bill amount
--greater than the average bill amount of all patients.

```
SELECT Name from Patients  
where BillAmount>  
(select avg(BillAmount)  
from Patients);
```

--Find the doctors who have a salary higher than
--the highest bill amount paid by any patient.

```
SELECT Name From Doctors  
where Salary > (select max(BillAmount) from Patients)
```

--List the patients who are in the same department as
--at least one doctor specializing in that department.

```
SELECT *  
FROM Patients P  
WHERE EXISTS (  
    SELECT 1  
    FROM Doctors D  
    WHERE D.Specialization = P.Department  
);
```



Union Family :

Union Family in databases refers to the use of the `UNION` operator to combine results from multiple queries that have the same structure. `UNION` removes duplicates, while `UNION ALL` allows duplicates in the final result.

```
USE HospitalDB;
```

--UNION all : must same number of columns and same datatype

```
SELECT Name FROM Doctors  
UNION all  
SELECT Name FROM Patients
```

--UNION : sorted and working distinct

```
SELECT Name FROM Doctors  
UNION  
SELECT Name FROM Patients
```

--intersect : give you the shared data

```
SELECT Name FROM Doctors  
intersect  
SELECT Name FROM Patients
```

```
SELECT Name FROM Doctors  
intersect  
SELECT Name FROM Patients
```

--except : the columns in first table that not in second table

```
SELECT Name FROM Doctors  
except  
SELECT Name FROM Patients
```



📌 What is EERD ?

EERD stands for **Enhanced Entity-Relationship Diagram**. It is an extension of the **ERD (Entity-Relationship Diagram)** that includes **advanced modeling concepts** to better represent complex database structures.

: ودا بنطبقو في الحالات التالية

✓ (مثل) عندما يكون لديك **علاقات وراثية** بين الكيانات **Employee** ← **Doctor** & **Nurse**).

✓ عندما تحتاج إلى **تصنيف الكيانات** بناءً على خصائص مشتركة.

✓ عندما تحتاج إلى **تمثيل أكثر دقة للعلاقات المعقدة** داخل قاعدة البيانات.

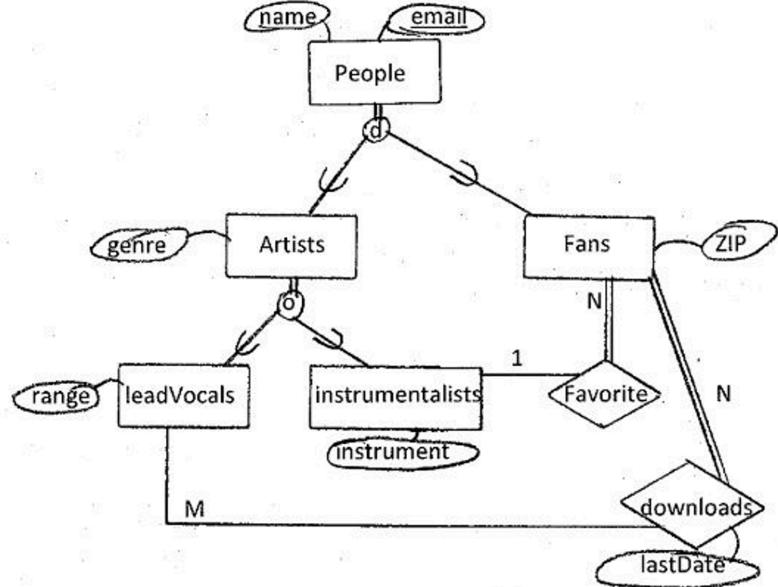
الفكرة ان لو عندك two entities من نفس النوع وفيهم بنיהם مشتركة فكل الي بنعملو زي فكرة استخدام الـ interface نعمل جدول يحتوي علي الي متكرر بنهم ااما الباقي الي المفروض هيورثو منهم هما الـ Sub Entities وهيبيقي الـ Super Entity ودا هيفدنا في اي ؟

✓ **تمثيل أفضل للواقع** عند تصميم قاعدة بيانات معقدة.

✓ **تحسين تنظيم البيانات** بتمثيل العلاقات المتداخلة والكيانات الهرمية.

✓ **أكثر مرونة** ويساعد في فهم العلاقة بين البيانات بشكل أعمق.

Example



◆ Super Entity

A **general entity** that contains **common attributes** shared by multiple sub-entities.

📌 **Example:** **Person** can be a **Student** or a **Teacher**, so **Person** is a **Super Entity**.

◆ Sub Entity

A **specialized entity** that inherits attributes from a **Super Entity** and adds its own unique attributes.

📌 **Example:** **Student** and **Teacher** are **Sub Entities** derived from **Person**.



what is a Constraints in Super type?

Completeness Constraints:

Total Specialization Rule: Yes (double line)

ودا يعني ان كل ال columns الي في ال super Entity موجودة في كل ال sub Entities

Partial Specialization Rule: No (single line)

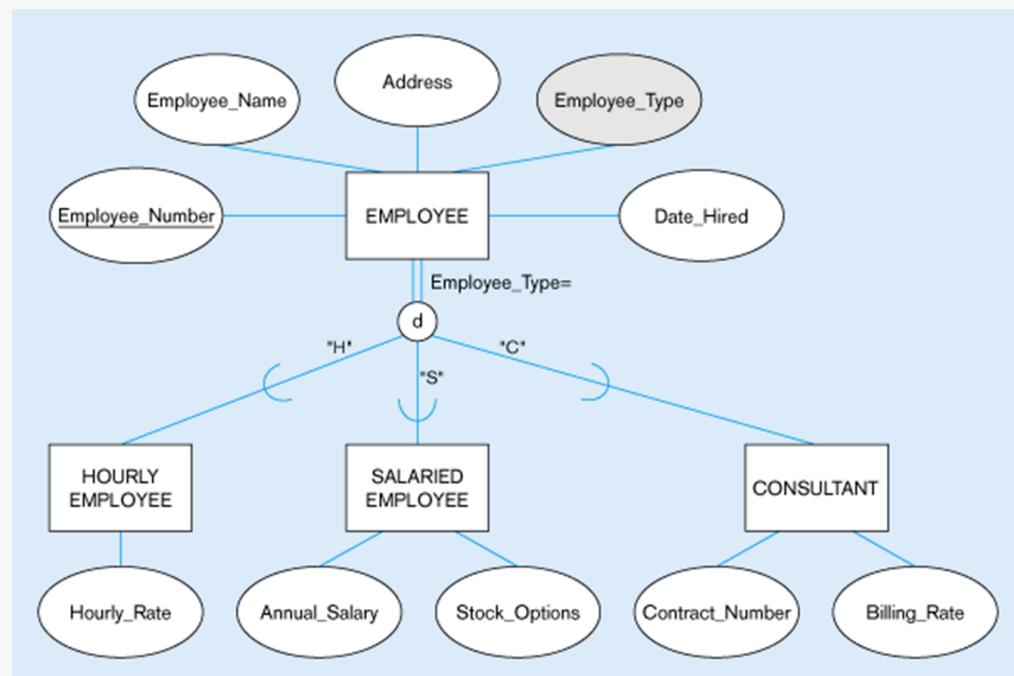
ودا يعني ان مش كل ال Columns الي في ال Super Entity موجودة في ال sub Entities

يعني عندك صفوف فالسوبر وملهاش تمثيل فال sub Entities

Disjointness Constraints

1. Total Disjoint Rule (d) - عدم التداخل التام

- في Sub Entity لا يمكن أن يكون جزءاً من أكثر من Super Entity هذا يعني أن الكيان في ال نفس الوقت.
- أي لا يمكن تكرار نفس الكيان في أكثر من Sub Entity.
- وألا داخل الدائرة عند العلاقة بين الـ (d) يتم تمثيله بـ Super Entity وـ Sub Entities.
- فممكن تضييف عمود جديد من عندك في السوبر وتسميه النوع كدا تحدد النوع الي هيشتغل لو حبيت تحدد بال ... Where type = ...



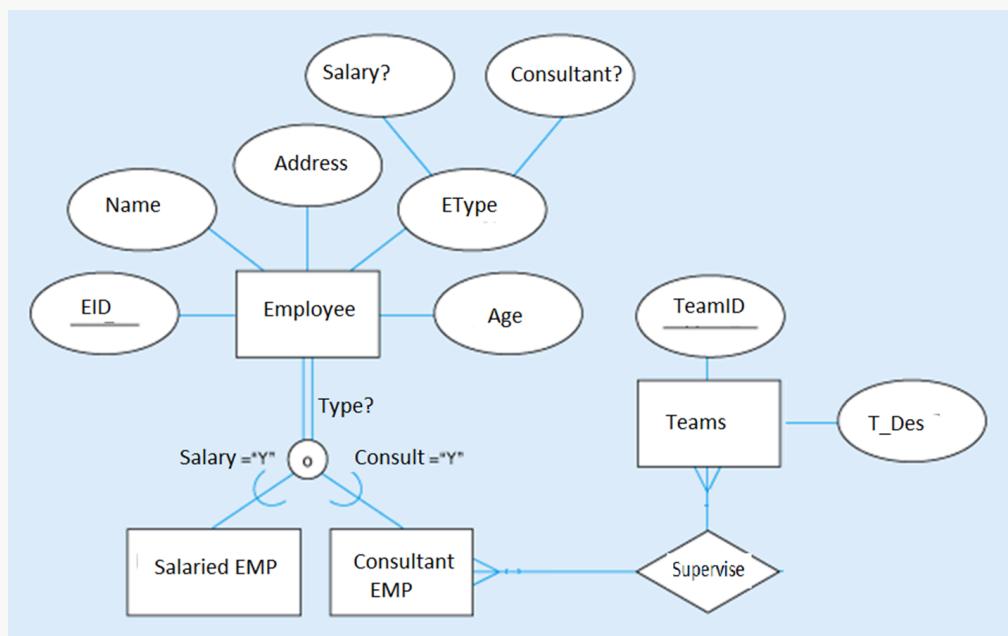
- مثال:

- "وله كيانات فرعية" مهندس (Employee) "إذا كان لدينا كيان "الموظف" فلا يمكن أن يكون نفس الموظف مهندساً ومحاسباً في نفس الوقت.

- ولكن لا يمكن أن يكون Employee يكون إما Engineer أو Accountant، بمعنى أن كل الاثنين معاً.

السماح بالتدخل - (o)

- في Sub Entity يمكن أن يكون جزءاً من أكثر من Super Entity هذا يعني أن الكيان في الا نفس الوقت.
- أي يمكن تكرار نفس الكيان في أكثر من Sub Entity.
- داخل الدائرة عند العلاقة بين الا (o) يتم تمثيله بـ Super Entity وال Sub Entities.
- هتعمل بحث لو عندك او معنديش composite Attribute



- مثال:

- "وله كيانات فرعية" مدرب (Trainer) "إذا كان لدينا كيان "الموظف" فمن الممكن أن يكون نفس الموظف مدرباً ومديراً في نفس الوقت.

الفرق بينهما:

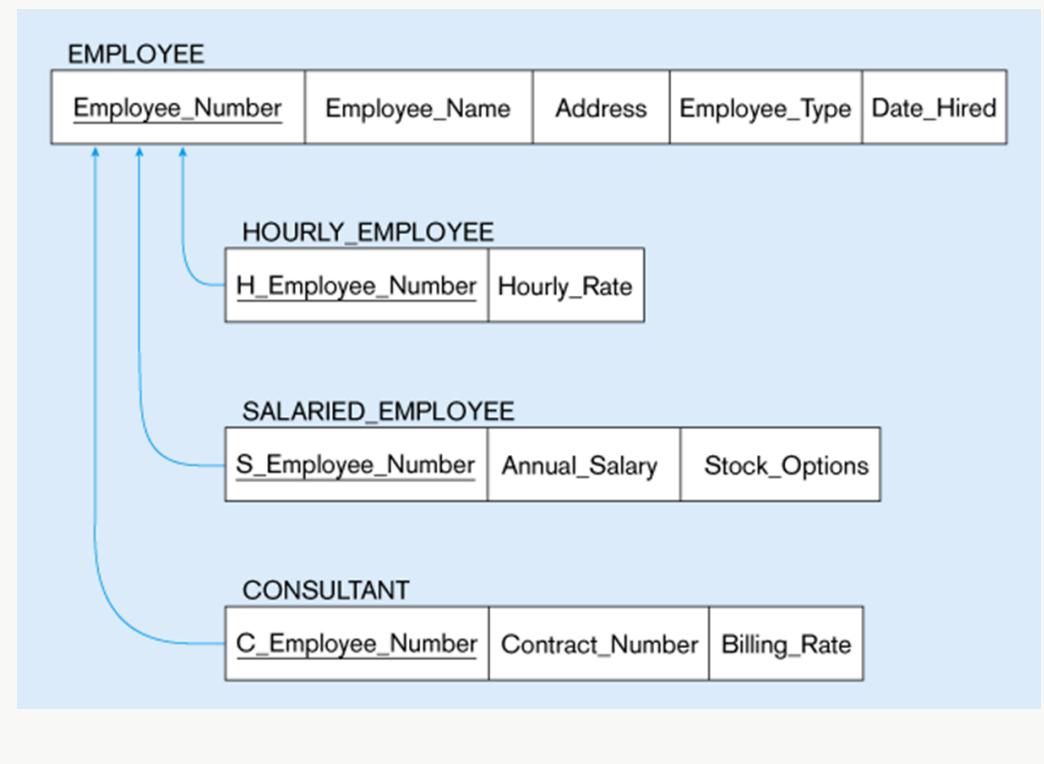
- الكيان لا يمكن أن يكون جزءاً من أكثر من كيان فرعى واحد (لا تكرار) \rightarrow Total Disjoint (d)
- الكيان يمكن أن يكون جزءاً من أكثر من كيان فرعى في نفس الوقت (يمكن التكرار) \rightarrow Overlap (o)



How can Map EERD ?

Create :

1. Table for Super Entity contain PK \Rightarrow SId
2. Table for every sub entity contain FK & PK from super entity





What happen when you install Microsoft SQL Server with all services ?

the instance installed that contain : **services and Applications**

services

1. DB Engine: SQL Server: (MSSQL Server)
2. SSIS ⇒ BI
3. SSRS⇒ BI
4. SSAS ⇒ BI
5. Data Quality Service

Applications

1. SQL Sever Management Studio ⇒ Working in DB Engine
2. SQL Server Data Tools ⇒ Working in BI
3. Data Quality Client ⇒ Working in Data Quality Service
4. SQL Server Profiler ⇒ Index
5. SQL Server Tuning Advisor ⇒Index



What is a types of Instance ?

1 Default Instance (الافتراضي)

- ♦ أول مرة SQL Server الأساسي الذي يتم إنشاؤه تلقائياً عند تثبيت Instance هو الـ.
- ♦ لا يحتاج إلى اسم مخصص، ويتم الوصول إليه مباشرةً باستخدام اسم السيرفر فقط.
- ♦ عند الاتصال به، يُستخدم:

SERVER_NAME

أو

(local)

أو

. (نقطة فقط).

مثال:

يمكنك الاتصال به هكذا، DESKTOP-12345 إذا كان اسم جهازك:

DESKTOP-12345

2 Named Instance (المسمى)

- ♦ على SQL Server إضافي يتم إنشاؤه عند الحاجة إلى أكثر من نسخة مستقلة من Instance هو نفس الجهاز.
- ♦ عند تثبيته، يجب تحديد اسم فريد له.
- ♦ يتم الاتصال به باستخدام الصيغة:

SERVER_NAME\INSTANCE_NAME

مثال:

فسسيكون الاتصال به، DESKTOP-12345 على جهازك Instance باسم SQL_Testing إذا أنشأست:

DESKTOP-12345\SQL_Testing



مقارنة سريعة بين Default و Named Instances

الميزة	Default Instance	Named Instance
الثبت	يتم تثبيته تلقائياً	يجب تحديده أثناء التثبيت
الاتصال	SERVER_NAME أو . (local)	SERVER_NAME\INSTANCE_NAME
إدارة الأداء	كل التطبيقات تعمل على نفس ال Instance	مختلفة Instances يمكن توزيع التطبيقات على نفس الجهاز
تشغيل إصدارات مختلفة	لا يدعم ذلك بسهولة	2022g على SQL Server 2016 يسمح بتشغيل نفس الجهاز
العزل الأمني	كل البيانات على نفس ال Instance	يمكن فصل قواعد البيانات والصلاحيات لكل Instance



Can You install same Engine many times ?

Yes

Use Cases

الحالة (Use Case)	لماذا تحتاج إلى أكثر من نسخة Engine؟	مثال عملي
تشغيل عدة إصدارات من SQL Server	بعض الأنظمة القديمة لا تعمل مع الإصدارات الأحدث، لذلك تحتاج إلى تشغيل إصدارات مختلفة.	لتطبيق قديم SQL Server 2016 تشغيل Server 2022 لتطبيق جديد.
عزل بيانات العمل (Development, Testing, Production)	لفصل قواعد البيانات الخاصة بالتطوير والاختبار والإنتاج، مما يمنع الأخطاء في البيانات الحقيقة.	للختبار وآخر للإنتاج حتى لا يتم Instance تشغيل للتعديل على بيانات العملاء بالخطأ.
تشغيل أكثر من Instance على نفس الجهاز	يعمل بشكل Instance كل مستقل، مما يسمح بعزل التطبيقات والموارد والصلاحيات.	للمحاسبة وآخر للإدارة Instance تشغيل المستشفى لتجنب تعارض البيانات.
استخدام أوضاع تشغيل مختلفة (Authentication Modes)	بعض التطبيقات تحتاج إلى Windows Authentication بينما أخرى تحتاج إلى SQL Authentication.	Instance تشغيل Windows Authentication بـ Instance الموظفي الشركة، و Authentication الويب لتطبيق الويب.
تشغيل قواعد بيانات حساسة بشكل منفصل	لتحسين الأمان عبر عزل البيانات الحساسة عن البيانات العامة.	لحسابات العملاء بيانات مشفرة، Instance تشغيل آخر للمحتوى العام.
تحسين الأداء عبر توزيع الحمل (Load Balancing)	لتقليل الضغط على واحد عبر توزيع التطبيقات على أكثر من Instance.	منفصل لكل تطبيق لمنع تعارض Instance تشغيل الاستعلامات الثقيلة.
إعداد بيئة اختبار مستقلة (Sandbox Environment)	لختبار التعديلات الجديدة على قاعدة بيانات منفصلة دون التأثير على الإنتاج.	مخصص للختبار لتجربة تحديثات Instance تشغيل النظام الجديدة.
التعامل مع تطبيقات تستخدم إعدادات متضاربة	بعض التطبيقات تحتاج إلى إعدادات مختلفة مثل حجم الـ TempDB أو خيارات الـ Collation.	وآخر بترميز UTF-8 بترميز Instance تشغيل SQL_Latin1_General_CI_AS.
استخدام إصدارات مختلفة من SQL Server من LocalDB	بعض التطبيقات تعتمد على إصدارات LocalDB محددة من لتتجنب مشكلات التوافق.	لتطبيق قديم LocalDB 2014 تشغيل LocalDB 2019 لتطبيق جديد.
إنشاء بيئة تجريبية للهجمات الأمنية (Penetration Testing)	لمحاكاة هجمات الاختراق على Instance دون المخاطرة بقاعدة البيانات الحقيقة.	مع تكوين ضعيف للختبار Instance تشغيل استراتيجيات الدفاع ضد الهجمات.
فصل البيانات لأغراض قانونية أو تنظيمية	بعض المؤسسات تتطلب عزل بيانات معينة بسبب قوانين GDPR.	منفصل لبيانات الاتحاد الأوروبي Instance تشغيل لضمان الامتثال للـ GDPR.

مثلك) الحماية GDPR).



Can You Use Different DB Engines at same App ?

Yes

Why?

❖ Database Engine	الغرض / الفائدة	💡 أمثلة على الاستخدام	🚀 كيفية التثبيت والاستخدام
Microsoft SQL Server (MSSQL)	المحرك الأساسي لإدارة البيانات العلائقية (Relational Database)	تخزين بيانات التطبيقات، التعامل مع الجداول والعلاقات، تشغيل الاستعلامات، إدارة المستخدمين	يتم تثبيته مع SQL Server مباشرة SSMS ويستخدم مع
Redis	تحسين الأداء عن طريق تخزين البيانات في الذاكرة (In-Memory Cache)	تخزين الكاش لتسريع تحميل البيانات، تخزين الجلسات في ASP.NET Core، تحسين أداء الاستعلامات المتكررة	أو تحميله Docker يمكن تثبيته عبر من الموقع الرسمي، واستخدامه مع StackExchange.Redis
Elasticsearch	محرك بحث متقدم لتحسين البحث داخل البيانات	البحث داخل النصوص الطويلة، البحث الفوري داخل بيانات المستخدمين، تحسين عمليات البحث في موقع التجارة الإلكترونية	أو يمكن تشغيله عبر Docker تثبيته، وربطه بـ ASP.NET Core باستخدام مكتبة NEST
MongoDB	NoSQL قاعدة بيانات لتخزين البيانات غير المنظمة (JSON-based Documents)	تخزين سجلات الدردشة، تسجيل الأحداث (Logs)، تخزين بيانات ديناميكية مثل الإعدادات	يمكن تشغيله عبر MongoDB أو تثبيته محلياً، وربطه بـ Atlas باستخدام مكتبة MongoDB.Driver
PostgreSQL	قاعدة بيانات قوية تدعم تحليل البيانات المتقدم والاستعلامات المعقدة	تحليل البيانات، التعامل مع JSON، تشغيل SQL، داخل عمليات إحصائية متقدمة	يمكن تثبيته من الموقع الرسمي أو باستخدام Docker، تشغيله عبر Npgsql في ASP.NET Core مع
Firebase Realtime Database	قاعدة بيانات سحابية لتحديث البيانات في الوقت الفعلي	تطبيقات الدردشة، الإشعارات الفورية، مزامنة بيانات المستخدمين بين الأجهزة	نُم Firebase تحتاج إلى حساب لربط Firebase SDK استخدام مكتبة التطبيق بها
MySQL / MariaDB	قاعدة بيانات مفتوحة المصدر تدعم الأنظمة الخفيفة	تشغيل موقع الويب الصغيرة، تخزين بيانات المدونات والمتأخر الإلكترونية	يمكن تثبيتها مباشرة أو تشغيلها عبر XAMPP / Docker وربطها MySql.Data باستخدام
SQLite	قاعدة بيانات خفيفة الوزن بدون خادم	تطبيقات سطح المكتب، تخزين بيانات محلية بدون الحاجة إلى خادم	يتم تثبيته تلقائياً مع بعض التطبيقات، ويمكن استخدامه مباشرة مع ASP.NET Core

	(Embedded Database)		
Oracle Database	محرك قواعد بيانات قوي للمؤسسات الكبيرة	تخزين بيانات الشركات الكبرى، تحليل البيانات الضخمة، تشغيل الأنظمة المصرفية	يمكن تثبيته محلياً أو تشغيله على السحابة، وربطه باستخدام Oracle.ManagedDataAccess
Cassandra	قاعدة بيانات NoSQL مصممة للتعامل مع البيانات الكبيرة (Big Data)	إدارة كميات هائلة من البيانات الموزعة، أنظمة التحليل الضخمة	يتم تثبيتها عبر Apache Cassandra أو Docker، وربطها باستخدام DataStax C# Driver
InfluxDB	قاعدة بيانات مخصصة لتخزين البيانات الزمنية (Time-Series Data)	تخزين بيانات الحساسات تحليل بيانات الخوادم، (IoT) تحليل أداء الأنظمة	يمكن تشغيلها عبر Docker، وربطها باستخدام InfluxDB.Client

✓ كيف تختار المدراك المناسب لمشروعك؟

- 🔳 تحتاج إلى قاعدة بيانات رئيسية لتخزين بيانات التطبيق؟ → استخدم **SQL Server**
- 🚀 تريدين سرعة تحميل البيانات؟ → استخدم **Redis**
- 🔎 تحتاج إلى بحث متقدم داخل النصوص؟ → استخدم **Elasticsearch**
- 📁 تحتاج إلى تخزين بيانات غير منتظمة؟ → استخدم **MongoDB**
- 📊 تحتاج إلى تحليلات بيانات متقدمة؟ → استخدم **PostgreSQL**
- 💬 تحتاج إلى تحديث البيانات في الوقت الفعلي؟ → استخدم **Firebase**
- 🛍️ تحتاج إلى قاعدة بيانات لمتجر إلكتروني صغير؟ → استخدم **MySQL**
- 📱 تحتاج إلى قاعدة بيانات خفيفة لتطبيق محمول؟ → استخدم **SQLite**
- 🏛️ تعمل في بيئة مؤسسية كبيرة؟ → استخدم **Oracle Database**
- 📈 تحتاج إلى تحليل بيانات كبيرة ومتغيرة بسرعة؟ → استخدم **Cassandra**
- ⏳ → استخدم (أو الخوادم IoT مثل) تحتاج إلى تسجيل البيانات الزمنية **InfluxDB**

◆ سيناريوهات لاستخدام أكثر من محرك قواعد بيانات في نفس المشروع

1 تحسين الأداء باستخدام Redis مع SQL Server

- يخزن البيانات في الذاكرة، مما يجعل الوصول إلى البيانات أسرع بكثير مقارنة بقاعدة Redis: لماذا؟ التقليدية SQL بيانات.
- كيف؟
 - قم ب تخزين البيانات الأكثر طلباً (مثل الجلسات، إعدادات المستخدم) في Redis.
 - لتخزين البيانات الدائمة مثل الحسابات والفوواتير SQL Server استخدم.
- بينما يتم حفظ بيانات Redis مثال: نظام تسجيل دخول سريع حيث يتم تخزين بيانات الجلسة في Redis المستخدمين في SQL Server.

2 تحسين البحث باستخدام Elasticsearch و SQL Server

- لماذا؟ SQL Server ليس قوياً في عمليات البحث المعقدة داخل النصوص الكبيرة، بينما مصمم لهذا الغرض.
- كيف؟
 - احفظ بيانات المستخدمين والمنتجات في SQL Server.
 - الأداء عمليات البحث السريع قم بإرسال نسخة من البيانات إلى Elasticsearch.
- بينما يتم تخزين Elasticsearch مثلاً: متر إلكتروني حيث يتم البحث عن المنتجات باستخدام تفاصيل المبيعات في SQL Server.

3 لتخزين البيانات غير المنظمة MongoDB مع SQL Server استخدام

- لماذا؟ بعض البيانات تكون ديناميكية وغير منتظمة (مثل الإعدادات أو البيانات الوصفية للمستخدمين).
- كيف؟
 - لتخزين البيانات الهيكلية مثل الحسابات والفواتير SQL Server استخدم MongoDB لتخزين البيانات غير المنظمة مثل الإشعارات أو سجل النشاطات (Logs).
 - بينما يتم SQL Server مثلاً: نظام يتعامل مع المستخدمين، حيث يتم تخزين بيانات الحساب في حفظ إعدادات المستخدم الشخصية والأنشطة في MongoDB.

4 لتحديث البيانات في الوقت الفعلي Firebase Realtime Database استخدام

- مصمم Firebase لا يدعم التحديثات اللحظية بدون استعلام مستمر، بينما SQL Server: لماذا؟ لهذا الغرض.
- كيف؟
 - احفظ بيانات المستخدمين في SQL Server.
 - لمزامنة البيانات مع واجهة المستخدم في الوقت الفعلي Firebase استخدم.
- ولكن يتم تحديث المحادثات في SQL Server مثلاً: تطبيق دردشة حيث يتم تخزين الرسائل في الوقت الفعلي Firebase.

5 لتحليلات البيانات مع PostgreSQL استخدام

- لديه دعم أفضل لتحليل البيانات المعقدة وعمليات الإحصاء المتقدمة مقارنة بـ PostgreSQL: لماذا؟ SQL Server.
- كيف؟
 - قواعد بيانات رئيسية PostgreSQL استخدم.
 - إنشاء التقارير والتحليلات PostgreSQL استخدم.
- ولكن يقوم بتحليل أداء SQL Server مثلاً: نظام موارد بشرية يقوم ب تخزين بيانات الموظفين في PostgreSQL.



What is a Difference Between Authentication and Authorization ?

Authentication ⇒ (Username , Password)

Authorization ⇒ Permission ⇒ انت مسؤولك تعمل اي بعد ما دخلت



What are the Types of Authentication in MSS?

Types of Authentication in Microsoft SQL Server (MSS)

Microsoft SQL Server supports **two main types** of authentication methods for controlling access:

1 Windows Authentication (Integrated Security)

- ◆ Uses **Windows user accounts** to authenticate users.
- ◆ Relies on **Active Directory (AD)** or local Windows users.
- ◆ No need to store passwords inside SQL Server.
- ◆ Most secure and recommended for enterprise environments.

Advantages:

- ✓ More secure (leverages Windows security policies).
- ✓ No need to manage separate SQL Server passwords.
- ✓ Supports **Single Sign-On (SSO)**.

Disadvantages:

- ✗ Only works for Windows-based environments.
- ✗ Cannot be used by non-Windows applications easily.

Example Connection String (SSMS & Applications)

```
Server=MY_SERVER; Database=MyDB; Integrated Security=True;
```

2 SQL Server Authentication

- ◆ Uses **username and password** stored in SQL Server.
- ◆ Independent of Windows login credentials.
- ◆ Required when clients **outside Active Directory** need access.

Advantages:

- ✓ Works across **Windows, Linux, and cloud** environments.
- ✓ Allows custom database users (without requiring Windows accounts).
- ✓ Useful for **web applications** that need separate DB credentials.

Disadvantages:

- ✗ Less secure (requires password storage & management).

 Needs **strong password policies** to prevent breaches.

 **Example Connection String (SSMS & Applications)**

```
Server=MY_SERVER; Database=MyDB; User Id=myUser; Password=myPassword;
```

Mixed Mode Authentication (Windows + SQL Authentication)

- ◆ Combines **both Windows Authentication & SQL Server Authentication**.
- ◆ Recommended when **some users are in Active Directory** and others are **external users**.

 **To enable Mixed Mode:**

- ✓ Open **SQL Server Management Studio (SSMS)**
- ✓ Right-click on the **Server → Properties → Security**
- ✓ Select **SQL Server and Windows Authentication Mode**

Summary Table

Authentication Type	Uses Windows Credentials?	Needs SQL Password?	Best For
Windows Authentication	 Yes	 No	Secure enterprise environments
SQL Server Authentication	 No	 Yes	Web apps, cross-platform access
Mixed Mode	 Yes &  No	 Yes &  No	Hybrid setups with internal & external users



What a Deference Between **TOP** & **TOP WITH TIES** ?

1 **TOP (N)**

- صَفَا فقط بناءً على الترتيب المحدد في **N** يقوم بإرجاع أول **ORDER BY**.
- إذا كان هناك صفوف متساوية في القيمة عند الحد الأقصى، فإنه لا يجلب الصفوف الإضافية.

مثال:

```
SELECT TOP 3 * FROM Employees ORDER BY Salary DESC;
```

- (هذا الاستعلام يجلب أعلى 3 موظفين) حسب الراتب.
- إذا كان هناك أكثر من 3 موظفين لديهم نفس الراتب في المركز الثالث، فلن يتم جلبتهم.

2 **TOP (N) WITH TIES**

- ولكنه يجلب أي صفوف إضافية تتساوى في الترتيب مع الصنف الأخير المختار **TOP** يشبهه.
- مفيد عند التعامل مع بيانات تحتوي على قيم متساوية وترغب في إرجاع جميع القيم المتساوية في الحد الأقصى.

مثال باستخدام WITH TIES :

```
SELECT TOP 3 WITH TIES * FROM Employees ORDER BY Salary DESC;
```

- إذا كان هناك 5 موظفين بنفس الراتب في المركز الثالث، فسيتم جلبتهم جميعاً.

مقارنة عملية

الميزة	TOP	TOP WITH TIES
يحدد عدد الصفوف بناءً على الترتيب	نعم <input checked="" type="checkbox"/>	نعم <input checked="" type="checkbox"/>
يجلب صفوف إضافية عند التساوي في الحد الأقصى	لا <input checked="" type="checkbox"/>	نعم <input checked="" type="checkbox"/>
ليعمل بشكل صحيح ORDER BY يحتاج	نعم <input checked="" type="checkbox"/>	نعم <input checked="" type="checkbox"/>

متى تستخدم **WITH TIES** ؟

- إذا كنت تريد ضمان عدم فقدان أي بيانات متساوية في التصنيف النهائي.
- عند إنشاء تقارير تتطلب العدالة في الترتيب، مثل أفضل 10 طلاب أو أعلى رواتب.
- صَفَا، لذا كن حذراً عند التعامل مع البيانات الكبيرة **N** قد يُرجع أكثر من **WITH TIES** لكن انتبه: استخدام **◆**.



How can order data random ?

✓ الطريقة الأساسية:

sql

CopyEdit

```
SELECT * FROM Employees ORDER BY NEWID();
```

◆ كيف يعمل؟

- لكل صف (GUID) يُنشئ معرّضاً فريداً.
- يتم ترتيب البيانات بشكل عشوائي في كل استعلام ، عند استخدامه في ORDER BY.

⌚ (TOP N) اختيار عدد عشوائي من الصفوف:

إذا كنت تزيد جلب عدد معين من الصفوف بشكل عشوائي، استخدم:

sql

CopyEdit

```
SELECT TOP 5 * FROM Employees ORDER BY NEWID();
```

◆ هذا الاستعلام يرجع 5 موظفين عشوائياً في كل مرة يتم تشغيله.

🎲 طريقة بديلة باستخدام (RAND())

sql

CopyEdit

```
SELECT * FROM Employees ORDER BY RAND();
```

▲ مشكلة RAND():

- يتم حساب قيمة واحدة فقط لكل الاستعلام، مما يؤدي إلى ترتيب غير متوقع.
- هو الخيار الأكثر استقراراً وعشوائية.



What is a Execution Order in SQL?

رقم	العبارة
1 FROM	تحديد الجداول التي سيتم جلب البيانات منها.
2 JOIN	دمج الجداول مع بعضها إذا كان هناك JOIN .
3 WHERE	(تصفية البيانات قبل أي عمليات تجميع GROUP BY).
4 GROUP BY	تجميع البيانات بناءً على عمود معين.
5 HAVING	تصفية البيانات بعد GROUP BY .
6 SELECT	تحديد الأعمدة المطلوبة وعرض النتائج.
7 DISTINCT	إزالة القيم المكررة إن وجدت.
8 ORDER BY	ترتيب البيانات حسب القيم المحددة.
9 TOP / OFFSET-FETCH	تحديد عدد الصفوف التي سيتم عرضها.

مثال عملي

استعلام مكتوب بهذه الطريقة

```
SELECT TOP 5 Department, COUNT(*) AS EmployeeCount
FROM Employees
WHERE Salary > 5000
GROUP BY Department
HAVING COUNT(*) > 3
ORDER BY EmployeeCount DESC;
```

ترتيب التنفيذ الفعلي

- 1 **FROM Employees** → استدعاء جدول **Employees**.
- 2 **WHERE Salary > 5000** → 5000 تصفية الموظفين ذوي الرواتب الأعلى من 5000.
- 3 **GROUP BY Department** → تجميع البيانات حسب **Department**.
- 4 **HAVING COUNT(*) > 3** → 3 تصفية الأقسام التي تحتوي على أكثر من 3 موظفين بعد التجميع.
- 5 **SELECT Department, COUNT(*) AS EmployeeCount** → اختيار الأعمدة المراد عرضها.
- 6 **ORDER BY EmployeeCount DESC** → ترتيب البيانات تناظرياً حسب عدد الموظفين.
- 7 **TOP 5** → اختيار أول 5 صفوف فقط.

لماذا هذا الترتيب مهم؟

- **WHERE** يتم تنفيذه قبل الأداء.
- **HAVING** يتم بعد التجميع لأنه يعمل على البيانات بعد التجميع.
- **ORDER BY** هو آخر خطوة تقريباً لأنها تعتمد على البيانات النهائية.
- **SELECT** يحتاج إلى معرفة البيانات النهائية قبل إرجاع SQL Server لأن الأعمدة المطلوبة.

❸ خلاصة

- ◆ ليس هو الترتيب الفعلي للتنفيذ SQL الترتيب المكتوب في.
- ◆ يتم تنفيذهما أولاً لتحديد البيانات و **WHERE**.
- ◆ يجمعان البيانات ويقومان بتصنيفتها **GROUP BY** ثم **HAVING**.
- ◆ يرتب النتائج **ORDER BY** يحدد الأعمدة المطلوبة، ثم
- ◆ يحددان عدد الصفوف المعروضة في النهاية **TOP** أو **OFFSET-FETCH**.



How can Copy table or slice of it in another table ?

by use **INTO**

```
--copy data and metadata
SELECT * INTO NewTable FROM OldTable;

--copy data only
INSERT INTO ExistingTable (Column1, Column2, Column3)
SELECT Column1, Column2, Column3 FROM SourceTable;
```



What is a Ranking Functions ?

1 ROW_NUMBER()

- ◆ يعطي رقمًا ترتيبياً لكل صف بدون تكرار، حتى لو كانت القيم مكررة.
- ◆ لا يكرر الأرقام عند وجود قيم متطابقة.
- ◆ يبدأ الترتيب من 1 ويزيد لكل صف.

◆ الصيغة:

```
SELECT
    EmployeeID, Name, Salary,
    ROW_NUMBER() OVER (ORDER BY Salary DESC) AS Rank
FROM Employees;
```

- ◆ إذا كان هناك رواتب متساوية، سيظل لكل موظف رقم مختلف.

2 RANK()

- ◆ يعطي نفس الترتيب للصفوف التي لها نفس القيم.
- ◆ لكن القيم التالية تخطى الأرقام (مثال: 1, 2, 2, 4).

◆ الصيغة:

```
SELECT
    EmployeeID, Name, Salary,
    RANK() OVER (ORDER BY Salary DESC) AS Rank
FROM Employees;
```

- ◆ إذا كان هناك موظفان برواتب متساوية، يحصلان على نفس الترتيب، لكن الرقم التالي يتخطى قيمة واحدة.

مثال عملي

EmployeeID	Name	Salary	Rank
1	Ahmed	9000	1
2	Sara	8000	2
3	Ali	8000	2
4	Mona	7000	4

3 DENSE_RANK()

- لكنه لا يتخطى الأرقام عند التكرار ، مشابه لـ `RANK()`.
- الرقم التالي دائمًا هو الرقم السابق + 1.

◆ الصيغة:

```
SELECT
    EmployeeID, Name, Salary,
    DENSE_RANK() OVER (ORDER BY Salary DESC) AS DenseRank
FROM Employees;
```

❖ مثال عملي

EmployeeID	Name	Salary	DenseRank
1	Ahmed	9000	1
2	Sara	8000	2
3	Ali	8000	2
4	Mona	7000	3

- يُقْعِدُ في الترتيب عند القيم المترادفة، بينما `RANK()` هو أن `DENSE_RANK()` يلاحظ الفرق بين `RANK()` و `DENSE_RANK()` بينما `RANK()` لا يفعل ذلك.

4 NTILE(N)

- مجموعات متساوية قدر الإمكان N يُقسّم البيانات إلى.
- كل مجموعة تأخذ رقمًا من 1 إلى N.
- مفید إذا كنت تريد تقسيم البيانات إلى أربع أو عشرات أو أي عدد آخر من الفئات.

◆ الصيغة:

```
SELECT
    EmployeeID, Name, Salary,
    NTILE(4) OVER (ORDER BY Salary DESC) AS Quartile
FROM Employees;
```

- هذا المثال يقسم البيانات إلى 4 مجموعات (أربع).

❖ مثال عملي (NTILE(4)):

EmployeeID	Name	Salary	Quartile
1	Ahmed	9000	1
2	Sara	8500	1

3	Ali	8000	2
4	Mona	7500	2
5	Hany	7000	3
6	Rania	6500	3
7	Omar	6000	4
8	Nour	5500	4

◆ كل ربع يحتوي على عدد متساوٍ من الصفوف تقريباً.

مقارنة بين الدوال الأربعه

الدالة	يكسر الأرقام؟	هل يتخطى الأرقام؟	المهمة الأساسية
ROW_NUMBER()	لا يكسر	نعم	يعطي رقمًا فريدياً لكل صف
RANK()	نعم	نعم	يعطي نفس الترتيب للقيم المتشابهة مع تخطي الأرقام
DENSE_RANK()	نعم	لا	يعطي نفس الترتيب للقيم المتشابهة دون تخطي الأرقام
NTILE(N)	لا يكسر	لا	مجموعات متساوية N يقسم البيانات إلى

متى تستخدم كل دالة؟

إذا كنت تريد ترقيم الصفوف بدون تكرار استخدم `ROW_NUMBER()`.

إذا كنت تريد إعطاء نفس الترتيب للصفوف المتكررة ولكن مع تخطي الأرقام استخدم `RANK()`.

إذا كنت تريد إعطاء نفس الترتيب للصفوف المتكررة دون تخطي الأرقام استخدم `DENSE_RANK()`.

إذا كنت تريد تقسيم البيانات إلى مجموعات متساوية `NTILE(N)` استخدم.

خلاصة

- يعطي أرقاماً فريدة بدون تكرار → `ROW_NUMBER()`.
- يعطي نفس الترتيب للصفوف المتكررة، لكن يتخطى الأرقام → `RANK()`.
- يعطي نفس الترتيب للصفوف المتكررة، لكن لا يتخطى الأرقام → `DENSE_RANK()`.
- مجموعات متساوية `N` يقسم البيانات إلى → `NTILE(N)`.

أي دالة تحتاج استخدامها؟ هل لديك مثال تريد تجربته؟



What is a PARTITION BY in SQL Server?

تُستخدم مع الدوال التحليلية (مثل دوال الترتيب، المجموع، المتوسط) لتقسيم البيانات إلى مجموعات، بحيث يتم تنفيذ العمليات داخل كل مجموعة بشكل منفصل.

❖ الفرق بين ORDER BY و PARTITION BY

- ORDER BY : يحدد ترتيب الصفوف عند جلب البيانات.
- PARTITION BY : يقسم البيانات إلى مجموعات قبل تطبيق الدالة عليها.

❖ أمثلة عملية على PARTITION BY

1 استخدام ROW_NUMBER() و PARTITION BY

```
SELECT
    Department, EmployeeID, Name, Salary,
    ROW_NUMBER() OVER (PARTITION BY Department ORDER BY Salary DESC) AS RowNum
FROM Employees;
```

ما الذي يحدث هنا؟

- يتم تقسيم الموظفين حسب القسم (Department).
- داخل كل قسم بشكل منفصل (RowNum) يتم إعطاء أرقام ترتيب.
- إذا كان لدينا 3 أقسام، فكل قسم سيبدأ العد من 1.

Department	EmployeeID	Name	Salary	RowNum
HR	1	Ahmed	9000	1
HR	2	Sara	8000	2
IT	3	Ali	9500	1
IT	4	Mona	8800	2

2 استخدام SUM() و PARTITION BY

```
SELECT
    Department, EmployeeID, Name, Salary,
    SUM(Salary) OVER (PARTITION BY Department) AS TotalSalary
FROM Employees;
```

ما الذي يحدث هنا؟

- يتم حساب مجموع الرواتب لكل قسم (**Department**).
- يتم إظهار مجموع الراتب داخل كل صف.

Department	EmployeeID	Name	Salary	TotalSalary
HR	1	Ahmed	9000	17000
HR	2	Sara	8000	17000
IT	3	Ali	9500	18300
IT	4	Mona	8800	18300

3 استخدام **AVG()** مع **PARTITION BY**

```
SELECT
    Department, EmployeeID, Name, Salary,
    AVG(Salary) OVER (PARTITION BY Department) AS AvgSalary
FROM Employees;
```

ما الذي يحدث هنا؟

- يتم حساب متوسط الرواتب لكل قسم (**Department**).
- يتم إظهار متوسط الراتب داخل كل صف.

Department	EmployeeID	Name	Salary	AvgSalary
HR	1	Ahmed	9000	8500
HR	2	Sara	8000	8500
IT	3	Ali	9500	9150
IT	4	Mona	8800	9150

4 استخدام **RANK()** مع **PARTITION BY**

```
SELECT
    Department, EmployeeID, Name, Salary,
    RANK() OVER (PARTITION BY Department ORDER BY Salary DESC) AS Rank
FROM Employees;
```

ما الذي يحدث هنا؟

- يتم إعطاء ترتيب داخل كل قسم (**Department**).
- يتم إعطاء نفس الترتيب للموظفين ذوي الرواتب المتساوية.

Department	EmployeeID	Name	Salary	Rank
HR	1	Ahmed	9000	1

HR	2	Sara	8000	2
IT	3	Ali	9500	1
IT	4	Mona	8800	2

⌚ متى تستخدم PARTITION BY ؟

✓ عند الحاجة إلى تقسيم البيانات إلى مجموعات دون تجميعها بالكامل.

✓ عند استخدام الدوال التحليلية (SUM , AVG , RANK , ROW_NUMBER).

✓ عند الحاجة إلى تشغيل دالة معينة على مجموعة بيانات منفصلة دون التأثير على البيانات الأخرى.



📌 Table of SQL Data Types

Category	Data Type	Description	Storage Size
Integer Numbers	TINYINT	Stores whole numbers from 0 to 255	1 byte
	SMALLINT	Stores whole numbers from -32,768 to 32,767	2 bytes
	INT	Stores whole numbers from -2,147,483,648 to 2,147,483,647	4 bytes
	BIGINT	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	8 bytes
Decimal Numbers	DECIMAL(p, s) / NUMERIC(p, s)	Fixed precision numbers (p = total digits, s = decimal places)	5-17 bytes
	FLOAT(n)	Approximate floating-point numbers (precision depends on n)	4 or 8 bytes
	REAL	Approximate floating-point numbers (lower precision than FLOAT)	4 bytes
Monetary Values	MONEY	Stores currency values (-922,337,203,685,477.5808 to 922,337,203,685,477.5807)	8 bytes
	SMALLMONEY	Stores smaller currency values (-214,748.3648 to 214,748.3647)	4 bytes
Date & Time	DATE	Stores only the date (YYYY-MM-DD)	3 bytes
	TIME	Stores only the time (hh:mm:ss)	3-5 bytes
	DATETIME	Stores both date and time (YYYY-MM-DD hh:mm:ss)	8 bytes
	SMALLDATETIME	Stores date & time but with lower precision	4 bytes
	DATETIME2(n)	Stores date & time with higher precision	6-8 bytes
String (Text) Types	DATETIMEOFFSET	Stores date & time with timezone offset	8-10 bytes
	CHAR(n)	Fixed-length string (max 8,000 characters)	n bytes
	VARCHAR(n)	Variable-length string (max 8,000 characters)	n + 2 bytes
Unicode Strings	TEXT	Large variable-length text (deprecated, use VARCHAR(MAX))	Up to 2GB
	NCHAR(n)	Fixed-length Unicode string (max 4,000 characters)	2 × n bytes

	<code>NVARCHAR(n)</code>	Variable-length Unicode string (max 4,000 characters)	2 × (n + 2) bytes
	<code>NTEXT</code>	Large Unicode text (deprecated, use NVARCHAR(MAX))	Up to 2GB
Binary Data	<code>BINARY(n)</code>	Fixed-length binary data	n bytes
	<code>VARBINARY(n)</code>	Variable-length binary data	n + 2 bytes
	<code>IMAGE</code>	Large binary objects (deprecated, use VARBINARY(MAX))	Up to 2GB
Boolean Type	<code>BIT</code>	Stores 0 (false), 1 (true), or NULL	1 bit per row
Unique Identifier	<code>UNIQUEIDENTIFIER</code>	Stores a GUID (Globally Unique Identifier)	16 bytes
Spatial Data	<code>GEOMETRY</code>	Stores geometric data (points, lines, polygons)	Variable
	<code>GEOGRAPHY</code>	Stores geographical data (latitudes, longitudes)	Variable
XML Data	<code>XML</code>	Stores XML-formatted data	Variable
JSON Data	<code>NVARCHAR(MAX)</code>	Stores JSON-formatted data (SQL Server does not have a native BOOLEAN type.)	Up to 2GB

✓ Notes

- `VARCHAR(MAX)` , `NVARCHAR(MAX)` , `VARBINARY(MAX)` → Can store up to **2GB** of data.
- `TEXT` , `NTEXT` , `IMAGE` → **Deprecated**, use `VARCHAR(MAX)` , `NVARCHAR(MAX)` , or `VARBINARY(MAX)` .
- `BIT` → Used for **boolean** values, even though SQL Server **does not have a native BOOLEAN type**.



What is deference between Case , IIF ?

1. CASE Statement

لتطبيق شروط متعددة وتحديد القيم بناءً على تلك الشروط CASE تستخدم.

مثال على CASE :

```
SELECT ProductName,
       UnitsInStock,
       CASE
           WHEN UnitsInStock = 0 THEN 'Out of Stock'
           WHEN UnitsInStock < 10 THEN 'Low Stock'
           ELSE 'In Stock'
       END AS StockStatus
FROM Products;
```

❖ متى نستخدم CASE ؟

- عندما يكون هناك أكثر من شرط واحد.
- عند الحاجة إلى تنفيذ شروط معقدة.

2. IIF Function متوفرة في SQL Server 2012+)

(في لغات البرمجة IF مثل) ولكنها تدعم شرطاً واحداً فقط CASE اختصاراً لـ IIF تعتبر.

مثال على IIF :

```
SELECT ProductName,
       UnitsInStock,
       IIF(UnitsInStock = 0, 'Out of Stock', 'Available') AS StockStatus
FROM Products;
```

❖ متى نستخدم IIF ؟

- عندما يكون هناك شرط واحد فقط مع قيمتين فقط (True/False).
- لجعل الكود أكثر اختصاراً وأسهل في القراءة.

مقارنة بين CASE و IIF

المقارنة	CASE	IIF
عدد الشروط	يدعم أكثر من شرط	يدعم شرطاً واحداً فقط

التعقيد	مناسب للحالات المعقدة	مناسب للحالات البسيطة
التوافق	متاح في جميع إصدارات SQL Server	متاح فقط من SQL Server 2012+
القابلية للقراءة	قد يكون أطول لكنه أكثر مرونة	أكثر اختصاراً لكنه أقل مرونة

متى تختار كل واحد؟

- ✓ إذا كان لديك **عدة شروط** أو تزيد مزيجاً من المرونة **CASE** استخدم.
- ✓ إذا كنت تحتاج فقط إلى شرط واحد مع حالتين **IIF** استخدم **(True/False)**.



What is Difference Between Cast ,Convert and Format ?

1. CAST

القياسية SQL تستخدم لتحويل البيانات من نوع إلى آخر وفقاً لمعايير

الأكثرون توافق مع معايير ANSI SQL

بسيطة وسهلة القراءة

مثال: تحويل رقم إلى نص

```
SELECT CAST(123 AS VARCHAR(10)) AS ConvertedValue;
```

◆ النتيجة: '123' (كتاب)

مثال: تحويل نص إلى رقم

```
SELECT CAST('123' AS INT) AS ConvertedValue;
```

◆ النتيجة: 123 (عدد صحيح)

2. CONVERT

لكنه يوفر تحكمًا إضافيًّا في التنسيقات، خاصة عند التعامل مع التواريخ والأرقام يشبهه.

يدعم تنسيقات إضافية للتاريخ والأرقام

متاح فقط في SQL Server

مثال: تحويل رقم إلى نص

```
SELECT CONVERT(VARCHAR(10), 123) AS ConvertedValue;
```

◆ النتيجة: '123' (كتاب)

مثال: تحويل تاريخ إلى نص بتنسيق معين

```
SELECT CONVERT(VARCHAR(10), GETDATE(), 103) AS FormattedDate;
```

◆ النتيجة: '21/02/2025' (تنسيق يوم/شهر/سنة)

📌 رقم 103 هو كود تنسيق (DD/MM/YYYY)

3. FORMAT

تُستخدم لتدوين القيم إلى نصوص مع دعم تخصيص تنسيقات التاريخ والأرقام بسهولة*.

✓ أسهل في التحكم بتنسيقات النصوص

✓ (مثلاً) يدعم اللغات والثقافات المختلفة en-US , ar-EG)

✗ أداء أبطأ مقارنة بـ CONVERT و CAST

مثال: تنسيق التاريخ بتنسيق معين

```
SELECT FORMAT(GETDATE(), 'dd/MM/yyyy') AS FormattedDate;
```

◆ النتيجة: 21/02/2025

مثال: تنسيق الأرقام مع الفواصل العشرية

```
SELECT FORMAT(1234567.89, 'N2') AS FormattedNumber;
```

◆ النتيجة: 1,234,567.89

🔍 مقارنة بين CAST و CONVERT و FORMAT

الخاصية	CAST	CONVERT	FORMAT
توافق مع ANSI SQL	نعم ✓	لا ✗	لا ✗
تحويل البيانات بين الأنواع	نعم ✓	نعم ✓	لا (فقط لتنسيق البيانات كنص) ✗
تحكم إضافي في التنسيق	لا ✗	نعم (يدعم أكواد التنسيق) ✓	نعم (يدعم أنماط مختصرة) ✓
دعم اللغات والثقافات	لا ✗	لا ✗	نعم ✓
الأداء	سرريع ⚡	سرريع ⚡	أبطأ 🐢

متى تستخدم كل واحد؟

القياسي SQL إذا كنت تحتاج إلى تحويل بيانات بسيط وسريع ومتواافق مع CAST استخدم.

إذا كنت تحتاج إلى تحكم إضافي في تنسيق التاريخ والرقم استخدم CONVERT.

إذا كنت تحتاج إلى تنسيق بيانات كنصوص بتفاصيل دقيقة، لكنك حذراً من الأداء الأبطأ استخدم FORMAT.

الخلاصة:

- هو الخيار الأكثر عمومية وبسيط للاستخدام.
- مفید عندما تحتاج إلى التحكم في تنسيق البيانات، خاصة مع التواریخ.
- هو الأفضل عند التعامل مع التنسيقات المتقدمة ولكن له أداء أقل مقارنة بالآخرين.



How can working with date ?

1 دوال جلب السنة، الشهر، اليوم

الدالة	الوصف	مثال	النتيجة
YEAR(date)	جلب السنة من التاريخ	SELECT YEAR(GETDATE());	2025
MONTH(date)	جلب الشهر من التاريخ	SELECT MONTH(GETDATE());	2
DAY(date)	جلب اليوم من التاريخ	SELECT DAY(GETDATE());	21

2 دوال جلب أجزاء أخرى من التاريخ

الدالة	الوصف	مثال	النتيجة
DATEPART(part, date)	جلب جزء معين من التاريخ	SELECT DATEPART(WEEKDAY, GETDATE());	4 (الأربعاء)
DATENAME(part, date)	جلب اسم الجزء كنص	SELECT DATENAME(MONTH, GETDATE());	'February'
DATEFROMPARTS(year, month, day)	إنشاء تاريخ معين	SELECT DATEFROMPARTS(2025, 2, 21);	2025-02-21

❖ **DATEPART** يُستخدم لجلب النصوص **DATENAME** بينما يُستخدم لجلب أرقام، بينما

❖ **الأجزاء المتاحة لـ DATEPART و DATENAME :**

- **YEAR** (السنة)
- **MONTH** (الشهر)
- **DAY** (اليوم)
- **WEEKDAY** (رقم اليوم في الأسبوع)
- **HOUR** (الساعة)
- **MINUTE** (الدقيقة)
- **SECOND** (الثانية)

3 دوال جلب التاريخ والوقت الحالي

الدالة	الوصف	مثال	النتيجة
GETDATE()	جلب التاريخ والوقت الحالي	SELECT GETDATE();	2025-02-21 14:30:00.000
SYSDATETIME()	جلب التاريخ والوقت الحالي بدقة أعلى	SELECT SYSDATETIME();	2025-02-21 14:30:00.1234567
CURRENT_TIMESTAMP	جلب التاريخ والوقت الحالي	SELECT CURRENT_TIMESTAMP;	نفس GETDATE()

❖ `SYSDATETIME()` أكثر دقة من `GETDATE()` لأنه يدعم `DATETIME2`.

4 دوال تعديل التاريخ

الدالة	الوصف	مثال	النتيجة
<code>DATEADD(part, value, date)</code>	إضافة عدد معين إلى جزء من التاريخ	<code>SELECT DATEADD(DAY, 5, GETDATE());</code>	2025-02-26
<code>DATEDIFF(part, start, end)</code>	حساب الفرق بين تاريخين	<code>SELECT DATEDIFF(DAY, '2025-01-01', GETDATE());</code>	51
<code>EOMONTH(date, add_months)</code>	جلب آخر يوم في الشهر	<code>SELECT EOMONTH(GETDATE());</code>	2025-02-28

❖ `DATEADD` لإضافة أو طرح قيمة من التاريخ.

❖ `DATEDIFF` لحساب الفرق بين تاريخين.

❖ `EOMONTH` لمعرفة آخر يوم في الشهر.

5 دوال تنسيق التاريخ

الدالة	الوصف	مثال	النتيجة
<code>FORMAT(date, format)</code>	تنسيق التاريخ كنص	<code>SELECT FORMAT(GETDATE(), 'dd/MM/yyyy');</code>	21/02/2025
<code>CONVERT(varchar, date, style)</code>	تحويل التاريخ إلى نص بنمط معين	<code>SELECT CONVERT(VARCHAR, GETDATE(), 103);</code>	21/02/2025

❖ `CONVERT` أكواد الشائعة:

- 101 → MM/DD/YYYY
- 103 → DD/MM/YYYY
- 120 → YYYY-MM-DD HH:MI:SS
- 126 → YYYY-MM-DDTHH:MI:SS (ISO)

الخلاصة

لجلب السنة، الشهر، اليوم.

لجلب أجزاء التاريخ بالأرقام أو النصوص.

لجلب التاريخ والوقت الحالي.

لتعديل وحساب الفروقات في التاريخ.

لتنسيق التاريخ كنص.



💡 EOMONTH

- ◆ `EOMONTH` آخر يوم في الشهر لأي تاريخ تختاره.
- ◆ يمكنك تحديد عدد الأشهر التي تريد التقدم أو الرجوع إليها.

✓ أمثلة بسيطة

1 آخر يوم في الشهر الحالي

```
SELECT EOMONTH(GETDATE());
```

- ◆ النتيجة: (آخر يوم في فبراير 2025) 28-02-2025

2 آخر يوم لشهر معين

```
SELECT EOMONTH('2025-06-15');
```

- ◆ النتيجة: 30-06-2025

3 آخر يوم بعد شهرين من الآن

```
SELECT EOMONTH(GETDATE(), 2);
```

- ◆ النتيجة: (آخر يوم في أبريل 2025) 30-04-2025

4 آخر يوم قبل 3 أشهر

```
SELECT EOMONTH(GETDATE(), -3);
```

- ◆ النتيجة: (آخر يوم في نوفمبر 2024) 30-11-2024

⌚ الخلاصة

- ✓ `EOMONTH(date)` → يعطيك آخر يوم في نفس الشهر.
- ✓ `EOMONTH(date, n)` → آخر يوم بعد `n` أشهر.
- ✓ `EOMONTH(date, -n)` → آخر يوم قبل `n` أشهر.



what is a DB Representations?

1 التمثيل المادي (Physical Representation)

هو الطريقة التي تخزن بها البيانات فعلياً على القرص الصلب، ويكون من:

◆ ملفات قاعدة البيانات الفعلية (Database Files)

1. **MDF (Main Data File)** (الملف الأساسي الذي يحتوي على الجداول، الفهارس، والبيانات) ↗ الفعلية.
 2. **NDF (Secondary Data File)** (ملفات بيانات إضافية تُستخدم عند الحاجة لتوزيع البيانات على) ↗ أكثر من قرص أو تحسين الأداء.
 3. **LDF (Log Data File)** (ملف السجلات الذي يخزن جميع العمليات التي تمت على قاعدة البيانات) ↗ لغرض الاستعادة (Recovery).
- ☞ البيانات داخل هذه الملفات تخزن في وحدات أصغر تسمى "صفحات بيانات" (Data Pages)، وحجمها 8 KB.

2 التمثيل المنطقي (Logical Representation)

هو الطريقة التي تنظم بها البيانات داخل قاعدة البيانات دون التطرق إلى كيفية تخزينها فعلياً، ويكون من:

◆ مجموعات الملفات (File Groups)

- هي طريقة لـ**لتحصيل ملفات البيانات** داخل قاعدة البيانات لتنظيم التخزين وإدارته بشكل أفضل.
- محدد بدلاً من وضع كل شيء في نفس الملف **File Group** يمكن وضع جداول معينة في.
- هناك نوعان رئيسيان:

1. **Primary File Group** ↗ MDF.

إضافية ↗ يمكن إنشاؤها لوضع جداول أو بيانات معينة في أماكن منفصلة.

☞ فائدة File Groups:

- توزيع البيانات على أكثر من قرص لتحسين الأداء.
- تسهيل إدارة البيانات من خلال التحكم في مكان تخزين الجداول والفهارس.

القاعدة العامة التي يجب أن تفهمها

تخزن على القرص الصلب (MDF, NDF, LDF) = **(Physical) التمثيل المادي**.

تنظم البيانات داخل قاعدة البيانات (File Groups) = **(Logical) التمثيل المنطقي**.

كل قاعدة بيانات يجب أن تحتوي على على الأقل

- ملف بيانات أساسى (MDF).
- ملف سجلات (LDF).
- ملف مجموعة أساسية (Primary File Group).

4. منفصلة مع NDF عند الحاجة إلى تخزين بيانات خدمة أو تحسين الأداء، يمكن استخدام

اذن لتحسين الاداء والتعامل بشكل parallel بدلا من sequential والاستفادة من ال

استخدم اكثر من File Group



How can control Delete and update in relational tables ?

1 CASCADE (التأثير التابع)

- عند حذف أو تحديث سجل في الجدول الأساسي، يتم حذف أو تحديث كل السجلات المرتبطة تلقائياً.
- هذا الخيار قد يكون خطيراً إذا لم يكن هناك تحكم دقيق في البيانات.

مثال:

```
CREATE TABLE Parent (
    ID INT PRIMARY KEY
);

CREATE TABLE Child (
    ID INT PRIMARY KEY,
    ParentID INT FOREIGN KEY REFERENCES Parent(ID) ON DELETE CASCADE ON UPD
    ATE CASCADE
);
```

التأثير:

- المربطين به (Child) سيتم حذف جميع الأبناء، (Parent) إذا تم حذف الأب.
- إذا تم تغيير ParentID في Child سيتم تغيير معرف الأب.

2 NO ACTION (عدم اتخاذ إجراء)

- هذا هو السلوك الافتراضي.
- سيمنع العملية ويرجع خطأ SQL Server إذا حاولت حذف أو تحديث سجل مرتبط، فإن يجب حذف أو تعديل السجلات المرتبطة يدوياً أولاً.

مثال:

```
CREATE TABLE Child (
    ID INT PRIMARY KEY,
    ParentID INT FOREIGN KEY REFERENCES Parent(ID) ON DELETE NO ACTION ON UP
    DATE NO ACTION
);
```

التأثير:

- سيظهر خطأ ولن يتم الحذف ، إذا حاولت حذف سجل من **Parent** مرتبط ب **Child**.
- نفس الشيء عند محاولة تحديث المفتاح الأساسي في **Parent**.

3 SET NULL (تعيين القيمة المرتبطة إلى)

- NULL** عند حذف أو تحديث السجل في الجدول الأساسي، يتم تعيين المفاتيح الخارجية المرتبطة إلى بدلاً من حذفها أو تديثها.
- يجب أن يكون العمود القابل للتأثير قبل **NULL**.

مثال:

```
CREATE TABLE Child (
    ID INT PRIMARY KEY,
    ParentID INT NULL FOREIGN KEY REFERENCES Parent(ID) ON DELETE SET NULL O
    N UPDATE SET NULL
);
```

التأثير:

- بدلاً من حذفه **NULL** في **Parent** ستصبح **ParentID** في **Child**.
- إذا تم تعديل معرف **Parent.ID** في **Child** إلى **NULL**.

4 SET DEFAULT (تعيين القيمة الافتراضية)

- عند حذف أو تديث سجل مرتبط، يتم تعيين المفتاح الأجنبي إلى قيمة افتراضية محددة مسبقاً.
- يجب أن يحتوي العمود على قيمة افتراضية باستخدام **DEFAULT**.

مثال:

```
CREATE TABLE Child (
    ID INT PRIMARY KEY,
    ParentID INT DEFAULT 1 FOREIGN KEY REFERENCES Parent(ID) ON DELETE SET DE
    FAULT ON UPDATE SET DEFAULT
);
```

التأثير:

- إلى 1 (القيمة الافتراضية) في **ParentID** في **Child** سيتم تعيين ، إذا تم حذف سجل من **Parent**.
- إلى 1 في **ParentID** في **Child** سيتم تعيين ، إذا تم تديث **Parent.ID**.

 **متى نستخدم كل خيار؟**

ال الخيار	أفضل استخدام له
CASCADE	عند الحاجة إلى حذف أو تعديل كل البيانات المرتبطة تلقائياً (مثل حذف جميع الطلبات عند حذف العميل).
NO ACTION	عندما تحتاج إلى التحكم في الحذف والتحديثات لضمان عدم فقدان البيانات الهامة.
SET NULL	عندما تريد الاحتفاظ بالسجلات ولكن بدون ارتباط بالجدول الأساسي.
SET DEFAULT	عند حذف البيانات الأساسية NULL عندما تحتاج إلى تعيين قيمة بديلة بدلاً من

💡 مثال عملی متکامل

```

CREATE TABLE Customers (
    CustomerID INT PRIMARY KEY,
    Name NVARCHAR(100)
);

CREATE TABLE Orders (
    OrderID INT PRIMARY KEY,
    CustomerID INT FOREIGN KEY REFERENCES Customers(CustomerID) ON DELETE C
ASCADE ON UPDATE NO ACTION
);

```

- **ON DELETE CASCADE:** إذا حذفنا عميلاً، سيتم حذف جميع طلباته تلقائياً.
- **ON UPDATE NO ACTION:** إذا كان مستخدماً في `CustomerID` في `Orders`.

🚀 خلاصة القاعدة العامة

1. **CASCADE:** يحذف أو يعدل البيانات المرتبطة تلقائياً.
2. **NO ACTION:** يمنع الحذف أو التحديث إذا كان هناك بيانات مرتبطة.
3. **SET NULL:** بدلاً من حذفها أو تحريرها NULL يجعل القيم المرتبطة.
4. **SET DEFAULT:** يستبدل القيم المرتبطة بقيمة افتراضية بدلاً من حذفها أو تحريرها.



What is a Schema?

A **schema** is a **logical structure** in a database that groups related objects like **tables**, **views**, **stored procedures**, and **functions** under a common name. It helps organize and manage data efficiently.

Benefits of Using a Schema

- Better Organization** – Groups related tables and objects together.
- Improved Security** – Controls access to specific data by assigning permissions.
- Performance Optimization** – Speeds up queries by narrowing the search space.
- Easier Maintenance** – Allows updates and changes without affecting the entire database.
- Supports Multi-Tenancy** – Keeps data **separate for different users or clients** in shared databases.



1 Creating a Schema and Adding Tables

To create a new schema and add tables inside it:

```
-- Step 1: Create a new schema named 'Hospital'  
CREATE SCHEMA Hospital;  
  
-- Step 2: Create a table inside the 'Hospital' schema  
CREATE TABLE Hospital.Doctors (  
    DoctorID INT PRIMARY KEY,  
    Name NVARCHAR(100),  
    Specialization NVARCHAR(100)  
);  
  
-- Step 3: Create another table inside the 'Hospital' schema  
CREATE TABLE Hospital.Patients (  
    PatientID INT PRIMARY KEY,  
    Name NVARCHAR(100),  
    Age INT  
);
```

2 Moving a Table to a Different Schema

If you have a table in another schema (e.g., `dbo`) and want to move it to the `Hospital` schema:

```
-- Move 'Patients' table from 'dbo' to 'Hospital' schema  
ALTER SCHEMA Hospital TRANSFER dbo.Patients;  
  
-- Move 'Doctors' table from 'OldSchema' to 'Hospital' schema  
ALTER SCHEMA Hospital TRANSFER OldSchema.Doctors;
```

3 Viewing All Tables in a Specific Schema

To check which tables exist in a particular schema:

```
SELECT TABLE_NAME  
FROM INFORMATION_SCHEMA.TABLES  
WHERE TABLE_SCHEMA = 'Hospital';
```

4 Deleting a Schema (If Needed)

Before deleting a schema, you must first remove or transfer its tables:

```
-- Drop tables inside the schema first
```

```
DROP TABLE Hospital.Doctors;
```

```
DROP TABLE Hospital.Patients;
```

```
-- Now drop the schema
```

```
DROP SCHEMA Hospital;
```



what is a synonym ?

A **synonym** is a word that has the same or nearly the same meaning as another word. For example, synonyms of "simple" include "basic," "easy," and "plain."

In databases (DB), if you're asking about **synonyms in SQL**, they refer to **alternative names** for database objects like tables, views, or stored procedures. In **Oracle SQL**, for instance, a **synonym** allows you to create an alias for a table to simplify access.

Example in **Oracle SQL**:

```
CREATE SYNONYM my_table FOR schema_name.actual_table;
```

This lets you access `actual_table` using `my_table` instead.



What is deference between Drop , Delete and TRUNCATE ?

1. DELETE

- يستخدم لحذف **بيانات معينة** من الجدول بناءً على شرط (WHERE).
- يمكن استعادة البيانات إذا كان داخلاً **معاملة (Transaction)** ولم يتم **COMMIT**.
- مما يسمح بالاسترجاع ، (ملف السجلات) **Log File** يتم تسجيل العملية بالكامل في (ROLLBACK).
- خاصة عند حذف عدد كبير من السجلات ، الأداء أبطأ من **TRUNCATE**.

مثال:

```
DELETE FROM Patients WHERE Age > 60;
```

البيانات فقط تُحذف، الجدول يبقى كما هو.

2. TRUNCATE

- **غير مدعومة (WHERE)** يحذف **كل البيانات** من الجدول بدون شرط.
- في بعض الأنظمة مثل **ما لم يكن داخلاً معاملة** لا يمكن استرجاع البيانات بعد التنفيذ (SQL Server).
- بل فقط تسجيل أن الجدول تم تفريغه، مما يجعله **لا يتم تسجيل كل عملية حذف في أسرع من (DELETE)**.
- الجدول إلى حالته الأصلية (Index) يعيد فهرس.

مثال:

```
TRUNCATE TABLE Patients;
```

كل البيانات تُحذف ولكن الجدول يبقى بدون تغيير.

3. DROP

- **والقيود (Indexes)** **الفهارس (Schema)** يحذف **الجدول بالكامل**، بما في ذلك البنية (**Constraints**).
- لا يمكن استرجاع البيانات أو الجدول بعد التنفيذ، إلا باستخدام نسخة احتياطية.
- ولكن ليس بشكل تفصيلي لكل صفات **Log File** يتم تسجيل العملية في.

مثال:

```
DROP TABLE Patients;
```

الجدول وكل بياناته تخفي نهايتها

ملف السجلات (Log File) الفرق في

العملية	هل يتم تسجيل كل صف في Log File؟	هل يمكن استرجاع البيانات؟	السرعة
<code>DELETE</code>	نعم، يتم تسجيل كل صف محذوف	<input checked="" type="checkbox"/> نعم، باستخدام <code>ROLLBACK</code> إذا كان داخل معاملة	أبطأ عند حذف كميات كبيرة
<code>TRUNCATE</code>	لا، فقط يتم تسجيل عمليات التفريغ	<input checked="" type="checkbox"/> لا يمكن الاسترجاع إلا في بعض الأنظمة مع <code>TRANSACTION</code>	أسرع من <code>DELETE</code>
<code>DROP</code>	لا، فقط يتم تسجيل إزالة الجدول	<input checked="" type="checkbox"/> لا يمكن الاسترجاع	الأسرع لكنه يحذف الجدول

متى تستخدم كل أمر؟

- عندما تحتاج إلى حذف بيانات معينة مع إمكانية التراجع `DELETE` استخدم.
- عندما تريد مسح جميع البيانات بسرعة دون حذف الجدول `TRUNCATE` استخدم.
- عندما تحتاج إلى إزالة الجدول بالكامل من قاعدة البيانات `DROP` استخدم.



How can Delete column from table without deleting structure of it (metadata)?

`update table with set column = null`



what is a DB Integrity?

1 (Domain Integrity)

يضمن أن البيانات داخل العمود تتنبأ بقواعد المحددة لها.

◆ القيود (Constraints):

✓ يحدد نوع القيم المسموحة بها - **(Data Type)** نوع البيانات.

✓ تحديد قيمة افتراضية مثل - **(Default)** القيمة الافتراضية 'Cairo'.

✓ السماح أو منع القيم الفارغة **(NULL / NOT NULL)**.

✓ لتحديد قيمة محددة داخل العمود - **(Check Constraint)** القيود الشرطية.

◆ الكائنات (Objects):

✓ تحدد شروط إضافية للبيانات - **(Rules)** القواعد.

✓ تنفذ تعليمات تلقائية عند حدوث عمليات معينة - **(المحفزات) Triggers**.

2 (Entity Integrity)

داخل الجدول له هوية فريدة **(Row)** يضمن أن كل صف.

◆ القيود (Constraints):

✓ يمنع التكرار ويضمن أن كل صف له معرف فريد - **(Primary Key - PK)** المفتاح الأساسي.

✓ واحدة **(Unique Constraint)** القيود الفريدة يمنع التكرار لكن يسمح بقيمة.

◆ الكائنات (Objects):

✓ لتحسين أداء البحث - **(Indexes)** الفهرس.

✓ تنفذ عمليات عند إدراج أو تعديل البيانات - **(المحفزات) Triggers**.

3 (Referential Integrity)

يضمن أن العلاقات بين الجداول صحيحة ومتناهية.

◆ القيود (Constraints):

✓ يربط بين جداولين لضمان وجود العلاقات الصحيحة - **(Foreign Key - FK)** المفتاح الأجنبي.

◆ الكائنات (Objects):

✓ يمكن استخدامها لتنفيذ قواعد الأعمال عند التحديث أو الحذف - **(المحفزات) Triggers**.

🔥 خلاصة المفاهيم الأساسية

✓ يحدد نوع البيانات والقيم المسموحة في الأعمدة **(Domain Integrity)**: تكامل النطاق.

✓ يضمن أن كل صف له معرف فريد داخل الجدول **(Entity Integrity)**: تكامل الكيان.

✓ يحافظ على العلاقات بين الجداول **(Referential Integrity)**: التكامل المرجعي.



What is a Types of Constraints in DB?

نوع القيد	الوصف	مثال SQL
PRIMARY KEY	يحدد العمود الأساسي الذي يميز كل سجل بشكل فريد ولا يسمح بالقيمة المكررة أو NULL.	<code>id INT PRIMARY KEY</code>
FOREIGN KEY	يربط بين جداولين لضمان سلامة العلاقات بين البيانات.	<code>FOREIGN KEY (DoctorID) REFERENCES Doctors(ID)</code>
UNIQUE	يضمن عدم تكرار القيم في العمود المحدد، لكنه يسمح بالقيمة NULL إذا لم يكن PRIMARY KEY.	<code>email VARCHAR(255) UNIQUE</code>
CHECK	يفرض شرطاً معيناً على القيم داخل العمود.	<code>age INT CHECK (age >= 18)</code>
DEFAULT	يحدد قيمة افتراضية للعمود إذا لم يتم توفر قيمة عند الإدراج.	<code>status VARCHAR(20) DEFAULT 'Active'</code>
NOT NULL	يمنع الأعمدة من قبول القيم الفارغة (NULL).	<code>name VARCHAR(100) NOT NULL</code>
INDEX	يسهل البحث والاستعلام لكنه لا يفرض سلامة البيانات.	<code>CREATE INDEX idx_name ON Employees(name)</code>
TRIGGER (Custom Constraint)	تلقائياً عند حدوث SQL ينفذ كود INSERT مثل عمليات على الجدول أو UPDATE.	<code>CREATE TRIGGER prevent_duplicate_appointments</code>
STORED PROCEDURE (Custom Constraint)	محزن في قاعدة SQL ينفذ كود البيانات لضمان قواعد عمل محددة.	<code>CALL InsertEmployee('John', 5000)</code>
VIEW (Custom Constraint)	ينشئ "منظوراً" للبيانات يقيّد الوصول إلى سجلات معينة.	<code>CREATE VIEW FutureAppointments AS SELECT * FROM Appointments WHERE AppointmentDate > CURDATE();</code>



How can store a driven Attribute ?

look at `netsalary` in code

وبكدا هجدها من علي الها رد محسوبة وجاهزة

```
create table Emps
(
Id int primary key,
Name varchar,
sal int,
overtime int,
netsalary as(isnull(sal,0)+isnull(overtime,0)) persisted
)
```



How can Create a Composite PK?

```
CREATE TABLE PatientDoctor (
PatientId INT NOT NULL,
DoctorId INT NOT NULL,
AppointmentDate DATETIME NOT NULL,
PRIMARY KEY (PatientId, DoctorId) -- Composite PK
);
```



what a deference between :

Constraint c1 unique(salary),

Constraint c2 unique(overtime),

Constraint c3 unique(salary,overtime),

Answer :

c1 and c2 ⇒ here salary unique allow null one time and overtime unique allow null one time and two columns independence

c3 ⇒ here salary and overtime unique dependence ⇒ can not repeated

Ex ⇒ if salary =8000 and overtime = 14 then in another table can not repeated this values

together again ⇒ can not found salary =8000 and overtime = 14 again



What is CHECK Constraint ?

تعريف

يستخدم لتحديد شروط على القيم داخل العمود لضمان إدخال بيانات صحيحة.

مثال عند إنشاء الجدول

```
CREATE TABLE Patients (
    ID INT PRIMARY KEY,
    Age INT CHECK (Age >= 0 AND Age <= 120) -- العمر يجب أن يكون بين 0 و 120;
);
```

بعد إنشاء الجدول مثال إضافية

```
ALTER TABLE Patients
ADD CONSTRAINT CHK_Age CHECK (Age >= 0 AND Age <= 120);
```

حذف CHECK Constraint

```
ALTER TABLE Patients
DROP CONSTRAINT CHK_Age;
```

مع أكثر من عمود مثال على

```
CREATE TABLE Orders (
    OrderID INT PRIMARY KEY,
    Quantity INT,
    Price DECIMAL(10,2),
    CHECK (Quantity > 0 AND Price > 0) -- الكمية والسعر يجب أن يكونا أكبر من 0
);
```

مميزاته:

- يمنع إدخال بيانات غير صحيحة
- يقلل الحاجة إلى التحقق من البيانات برمجيًا



What happen if you alter table to add constrain in any columns ?

if data in columns not apply constrains it not created

and if you want to apply constrains in new data use Rule

else

it created



What is a Rule and how use it ?

A

Rule is a database object used to **enforce a condition** on the values entered into a column. It was mainly used in **older versions of SQL Server** but has been **replaced by CHECK constraints** because they are more flexible and easier to manage and it applied in new data and it created in database level

◆ مثال على إنشاء Rule

```
CREATE RULE AgeRule  
AS @Age >= 18 AND @Age <= 65;
```

◆ بعمود معين ربط Rule

```
sp_bindrule 'AgeRule', 'Employees.Age';
```

◆ من العمود إزالة Rule

```
sp_unbindrule 'Employees.Age';
```

◆ حذف Rule نهائياً

```
DROP RULE AgeRule;
```

Rule can be shared with many objects



What is a Shared Default ?

📌 Definition:

In **older versions of SQL Server** (before SQL Server 2008), the `CREATE DEFAULT` statement was used to **define a shared default value** that could be applied to multiple columns across different tables. This approach is now **deprecated** and replaced by `DEFAULT CONSTRAINT`.

◆ Creating a Shared DEFAULT

```
CREATE DEFAULT df1 AS 200; -- Defines a default value of 200
```

◆ Binding DEFAULT to a Column

```
sp_bindefault 'df1', 'Emps.Salary';
```

Now, if no value is provided for `Salary` in the `Emps` table, it will default to `200`.

◆ Unbinding the DEFAULT

```
sp_unbindefault 'Emps.Salary';
```

Removes the default value from the `Salary` column.

◆ Dropping the Shared DEFAULT

```
DROP DEFAULT df1;
```

Deletes the `df1` default so it can no longer be used.



How can create your own datatype ?

- Shared Data Type Constraints Using `sp_addtype`, `RULE`, and `DEFAULT` (Legacy SQL Server Methods)

Before **SQL Server 2008**, `sp_addtype`, `RULE`, and `DEFAULT` were used to create **custom data types with constraints and default values**.

These methods are **deprecated**, and Microsoft now recommends using **User-Defined Data Types** (`CREATE TYPE`) and `CHECK` constraints.

1 Create a Shared Data Type (`sp_addtype`)

```
EXEC sp_addtype 'dragon', 'INT';
```

- Creates a new custom data type named `dragon` of type `INT` .

2 Create a Rule (Validation Constraint)

```
CREATE RULE r1 AS (@x > 100);
```

- Ensures values assigned to `dragon` must be greater than 100.

3 Create a Shared Default Value

```
CREATE DEFAULT d1 AS 200;
```

- Defines `200` as the default value for `dragon` type.

4 Bind the Rule to the Data Type

```
sp_bindrule r1, 'dragon';
```

- Applies the rule (`r1`) to `dragon`, so all values must be greater than 100.

5 Bind the Default Value

```
sp_bindefault d1, 'dragon';
```

 Now, any column using `dragon` will have `200` as its default value if no value is provided.

6 Using the Shared Data Type in a Table

```
CREATE TABLE Employees (
    ID INT PRIMARY KEY,
    Salary dragon -- Uses the custom type with rule & default
);
```

 Now, `Salary` is forced to be `>100` and defaults to `200`.

7 Unbind & Drop the Constraints

```
sp_unbindrule 'dragon'; -- Removes rule
sp_unbindefault 'dragon'; -- Removes default
DROP RULE r1; -- Deletes the rule
DROP DEFAULT d1; -- Deletes the default
DROP TYPE dragon; -- Deletes the custom type
```



What is a types of Variable in SQL ?

1 Local Variables (`@variable`) – User-defined variables that exist only within a specific scope, such as a procedure or batch.

2 Global Variables (`@@variable`) – System-defined variables that store SQL Server environment information and are accessible globally.



What is a Local Variables?

Local variables in SQL Server are **declared** using the `DECLARE` statement and **initialized** using different methods.

1 Declare Without Initialization

```
DECLARE @x INT;
```

✓ Declares `@x` as an integer, but it has `NULL` as its value.

2 Declare and Initialize Using `SET`

```
DECLARE @x INT;  
SET @x = 10;
```

✓ First, declares `@x`, then assigns `10` using `SET`.

3 Declare and Initialize Using `SELECT`

```
DECLARE @x INT;  
SELECT @x = 20;
```

✓ Uses `SELECT` instead of `SET`.

🚀 `SELECT` is faster than `SET` for multiple assignments!

4 Declare and Initialize in One Step (Recommended ✓)

```
DECLARE @x INT = 30;
```

✓ Simpler syntax, combines declaration & initialization.

5 Declare Multiple Variables

```
DECLARE @a INT = 1, @b VARCHAR(50) = 'Hello', @c DATETIME = GETDATE();
```

✓ Declares & initializes multiple variables in one line.

6 Assign Values from a Query

```
DECLARE @Salary INT;
SELECT @Salary = Salary FROM Employees WHERE ID = 1;
```

 Assigns the `Salary` of `ID = 1` to `@Salary`.

Create a Local Table Variable & Insert Data Using `SELECT`

```
DECLARE @Employees TABLE (
    ID INT PRIMARY KEY,
    Name VARCHAR(50),
    Salary DECIMAL(10,2)
);

-- Insert data from another table
INSERT INTO @Employees (ID, Name, Salary)
SELECT EmployeeID, EmployeeName, EmployeeSalary FROM Employees WHERE Department = 'IT';

-- View inserted data
SELECT * FROM @Employees;
```

Key Points:

- The `@Employees` table stores temporary data.
- `INSERT INTO ... SELECT` pulls data from an existing table.
- Data exists only within the current batch or procedure.

Execute query from string (dynamic query)

```
declare @x nvarchar(50) ='*',@tb nvarchar(50) = 'Employees';
EXEC('SELECT '+@x + 'FROM ' + @tb)
```

Key Differences Between `SET` and `SELECT`

Feature	SET	SELECT
Works with multiple values?	✗ No (only one value)	✓ Yes (last row value)
Performance	🚀 Slightly slower	🔥 Faster for bulk operations
Standard SQL?	✓ Yes	✗ No

📌 Best Practices

- ✓ Use `DECLARE @var = value` for simple initialization.
- ✓ Use `SELECT` for bulk assignments, but be careful of multiple row results.
- ✓ Use `SET` when assigning a single value for clarity.



What is a Global variables ?

📌 Global variables (`@@variable`) are system-defined variables that store information about the SQL Server environment. They provide details like server status, configurations, and session-specific data.

◆ Examples of Global Variables:

- `@@VERSION` → Returns the SQL Server version.
- `@@ROWCOUNT` → Returns the number of rows affected by the last query.
- `@@IDENTITY` → Returns the last inserted identity value.
- `@@TRANCOUNT` → Shows the number of active transactions.

📌 Global variables cannot be modified by users; they are read-only.



Control Flow Statements in SQL Server

Control flow statements manage the logical flow of execution in SQL.

1 IF...ELSE (Conditional Execution)

📌 Executes different blocks based on a condition.

```
IF EXISTS (SELECT 1 FROM Employees WHERE Salary > 5000)
begin
    PRINT 'High Salary';
    End
ELSE
begin
    PRINT 'Low Salary';
    End

declare @x int
update Products
set UnitPrice+=1
select @x= @@ROWCOUNT
if(@x>0)
begin
    select 'multi rows affected'
    select 'kkaska'
    End
else
print 'no rows affected'

--to ask metadata that a table exists or not

if Exists(select name from sys.tables where name = 'Employees')
print 'Table founded'
else
print 'Table not found'
```

2 CASE (Inline Conditional Logic)

📌 Returns a value based on conditions.

```
SELECT Name,
CASE
    WHEN Salary > 5000 THEN 'High'
    ELSE 'Low'
END AS SalaryCategory
FROM Employees;
```

3 WHILE (Looping)

📌 Repeats execution while a condition is true.

```
DECLARE @count INT = 1;

WHILE @count <= 5
BEGIN
    PRINT 'Iteration: ' + CAST(@count AS NVARCHAR);
    SET @count = @count + 1;
END;
```

4 GOTO (Jump to a Label)

📌 Redirects execution to a specific label.

```
DECLARE @x INT = 10;

IF @x = 10
    GOTO MyLabel;

PRINT 'This will be skipped';

MyLabel:
PRINT 'Jumped here!';
```

5 TRY...CATCH (Error Handling)

📌 Handles exceptions in SQL.

```
BEGIN TRY
    INSERT INTO Employees (ID, Name) VALUES (1, 'John');
END TRY
BEGIN CATCH
    PRINT 'Error Occurred: ' + ERROR_MESSAGE();
    select Error_line()
END CATCH;
```



What a Deference between Batch & Script & Transaction ?

1. Batch (دفعة):

- يتم تنفيذها كدفعة واحدة SQL مجموعة من أوامر.
- يتم إرسالها إلى قاعدة البيانات دفعة واحدة، ولكن كل أمر يتم تنفيذه بشكل مستقل.
- في حال حدوث خطأ، قد يستمر تنفيذ الأوامر الأخرى ما لم يكن هناك تحكم إضافي مثل `TRY...CATCH` أو `TRANSACTION`.
- مثال في **SQL Server**:

```
UPDATE Patients SET Age = Age + 1 WHERE Age < 50;  
DELETE FROM Logs WHERE Date < '2023-01-01';
```

2. Script (نص برمجي):

- يتم تنفيذها معًا، مثل SQL ملف يحتوي على مجموعة من أوامر `.sql` في SQL Server.
- ويتم تنفيذه في بيئة قاعدة البيانات (**Batches**) يمكن أن يحتوي على **عدة دفعات**.
- يمكن أن يحتوي على تعليمات، متغيرات، وإجراءات مخزنة.
- يتم تنفيذه باستخدام أداة مثل **SQL Server Management Studio (SSMS)** أو **MySQL Workbench**.
- مثال على **Script**:

```
--إنشاء جدول جديد  
CREATE TABLE Doctors (  
    ID INT PRIMARY KEY,  
    Name VARCHAR(100),  
    Specialization VARCHAR(100)  
);  
  
--إدخال بيانات  
INSERT INTO Doctors (ID, Name, Specialization) VALUES (1, 'Dr. Ali', 'Ophthalmology');
```

3. Transaction (معاملة):

- مجموعة من العمليات التي يتم تنفيذها كوحدة واحدة، بحيث إذا فشل أحد الأوامر، يتم إلغاء جميع الأوامر الأخرى (`ROLLBACK`).

- يضمن سلامة البيانات ويستخدم لضمان **ACID (Atomicity, Consistency, Isolation, Durability)**.
- للغائتها لحفظ التغييرات، أو **COMMIT** يجب استخدام **ROLLBACK**.
- مثال في **SQL Server**:

```
Create Table Parent (ID int Primary key)
```

```
CREATE TABLE Child (PID INT FOREIGN KEY REFERENCES Parent(ID))
```

```
begin try
    begin transaction
        insert into Parent values (1)
        insert into Parent values (2)
        insert into Parent values (3)
        insert into Child values (1)
        insert into Child values (2)
        insert into Child values (5)
        commit
    end try
begin catch
    rollback
end catch

select * from parent -- no data
select * from Child -- no data
```

الفرق الجوهرى:

المصطلح	الوصف
Batch	يتم تنفيذها دفعة واحدة ولكن كل أمر بشكل مستقل SQL مجموعة أوامر.
Script	يمكن تشغيلها معاً SQL ملف يحتوي على مجموعة من أوامر.
Transaction	مجموعة من الأوامر تُنفذ كوحدة واحدة، وتلغى بالكامل إذا حدث خطأ.

أما إذا كنت تحتاج إلى تفريذ مجموعة من الأوامر ولكن تضمن سلامة البيانات، فاستخدم وإذا كنت تريد تنفيذ مجموعة أوامر متتالية دون التأكد من نجاحها جميعاً، يمكنك استخدام كنت تريد تنفيذ أوامر مخزنة في ملف، فاستخدم **Transaction**. **Batch**. **Script**.



What Happen When you execute transactional queries like `INSERT` , `DELETE` , or `UPDATE` , and the server restarts before the transaction is completed?

SQL Server follows ACID (Atomicity, Consistency, Isolation, Durability) principles to maintain data integrity. Here's what happens in different scenarios:

◆ Scenario 1: Server Restarts Before `COMMIT`

If the server crashes or restarts **before** the transaction is committed, **all changes are rolled back automatically** when the server starts again.

Example:

```
BEGIN TRANSACTION;
INSERT INTO Employees (ID, Name, Position) VALUES (1, 'Ali', 'Doctor');
DELETE FROM Patients WHERE ID = 5;
-- Server crashes before COMMIT
COMMIT;
```

◆ Since `COMMIT` was **not executed**, the transaction is **not saved**, and after the restart, SQL Server **rolls back** all changes.

◆ Scenario 2: Server Restarts After `COMMIT`

If the server restarts **after** `COMMIT` , the transaction is **permanently saved** in the database, even if the server crashes immediately.

Example:

```
BEGIN TRANSACTION;
INSERT INTO Employees (ID, Name, Position) VALUES (1, 'Ali', 'Doctor');
DELETE FROM Patients WHERE ID = 5;
COMMIT; -- Server crashes immediately after this
```

◆ Since `COMMIT` was executed, changes are **stored permanently**, and no rollback occurs after the restart.

◆ Scenario 3: Server Restarts During Execution (Before Reaching `COMMIT`)

If the server restarts **while the transaction is still executing**, SQL Server **rolls back** any partial changes when it starts again.

Example:

```
BEGIN TRANSACTION;  
INSERT INTO Employees (ID, Name, Position) VALUES (1, 'Ali', 'Doctor');  
-- Server crashes here before DELETE runs  
DELETE FROM Patients WHERE ID = 5;  
COMMIT;
```

- ◆ Since the `COMMIT` was not executed, the transaction **will be fully rolled back**, meaning the `INSERT` will not persist.

◆ How Does SQL Server Handle This?

SQL Server has a **Transaction Log** that tracks all changes. When the server restarts:

1. **It checks the transaction log** for incomplete transactions.
2. **If a transaction was not committed**, SQL Server **rolls it back**.
3. **If a transaction was committed before the crash**, the data remains **permanently stored**.



What is a Types of User Define Functions ?

1. Scalar Function : return one value
2. Inline Table Function :return table contain select statements only like View
3. Multi Statement Table Valued Function : return table contain select statements and another statements
in any type can not use Execute function in it



What is a Scalar Function and how declaration it and use ?

A **Scalar Function** is a **User-Defined Function (UDF)** that returns **a single value** (such as a number, text, or date) when called.

Example: A function that returns the square of a number

```
create function GetName(@id int)
returns varchar(10)
begin
declare @val varchar(10);
select @val =ProductName from Products
where ProductID = @id
return @val
End
```

```
select dbo.GetName(4) --must use dbo(schema) here
```



What is a Inline Table-Valued Function?

An **Inline Table-Valued Function (TVF)** is a **User-Defined Function (UDF)** that **returns a table** as a result. Unlike scalar functions, it does not use `BEGIN` and `END` but directly returns a query.

Example: A function that returns products by category

```
CREATE FUNCTION GetProductsByCategory(@CategoryID INT)
RETURNS TABLE
AS
RETURN
(
    SELECT ProductID, ProductName, Price
    FROM Products
    WHERE CategoryID = @CategoryID
);
```

Usage:

```
SELECT * FROM dbo.GetProductsByCategory(1);
```

Difference from Multi-Statement Table-Valued Function:

- **Inline TVF:** Directly returns a `SELECT` query (faster, optimized).
- **Multi-Statement TVF:** Uses `BEGIN...END` and inserts data into a table variable.



what is Multi-Statement Table-Valued Function ?

A **Multi-Statement Table-Valued Function (MSTVF)** is a **User-Defined Function (UDF)** that **returns a table** and allows multiple SQL statements inside the function body. Unlike an **Inline Table-Valued Function**, it uses a **table variable** and explicit **INSERT** statements.

✓ Example: MSTVF with IF condition

This function returns products based on price range, using an **IF** condition inside the function.

```
CREATE FUNCTION GetProductsByPrice(@PriceCategory VARCHAR(10))
RETURNS @ProductTable TABLE
(
    ProductID INT,
    ProductName VARCHAR(100),
    Price DECIMAL(10,2)
)
AS
BEGIN
    IF @PriceCategory = 'Low'
        BEGIN
            INSERT INTO @ProductTable
            SELECT ProductID, ProductName, Price
            FROM Products
            WHERE Price < 50;
        END
    ELSE IF @PriceCategory = 'High'
        BEGIN
            INSERT INTO @ProductTable
            SELECT ProductID, ProductName, Price
            FROM Products
            WHERE Price >= 50;
        END
    RETURN;
END;
```

📌 Usage:

```
SELECT * FROM dbo.GetProductsByPrice('Low'); -- Returns products with price < 50  
SELECT * FROM dbo.GetProductsByPrice('High'); -- Returns products with price >= 50
```



What is Windowing Functions?

لتحليل البيانات عبر مجموعة من الصيغ دون الدوال تُستخدم مع **Window Functions** في SQL هذه الدوال. وهي مفيدة في مقارنة القيم عبر الصيغ السابقة أو التالية للتأثير على عددها.

1. LAG() - جلب قيمة الصيغ السابق

تُستخدم لاسترجاع قيمة من الصيغ السابق بالنسبة للصيغ الحالي.

❖ **مثال:**

```
SELECT
    EmployeeID,
    Name,
    Salary,
    LAG(Salary, 1, 0) OVER(ORDER BY EmployeeID) AS PreviousSalary
FROM Employees;
```

هنا، `LAG(Salary, 1, 0)` تعني:

- العمود الذي سيتم إرجاع قيمته من الصيغ السابق → `Salary`.
- عدد الصيغ التي سيتم الرجوع إليها (صف واحد للخلف) → `1`.
- القيمة الافتراضية إذا لم يكن هناك صيغ سابق → `0`.

2. LEAD() - جلب قيمة الصيغ التالي

تُستخدم لاسترجاع قيمة من الصيغ التالي بالنسبة للصيغ الحالي.

❖ **مثال:**

```
SELECT
    EmployeeID,
    Name,
    Salary,
    LEAD(Salary, 1, 0) OVER(ORDER BY EmployeeID) AS NextSalary
FROM Employees;
```

هنا، `LEAD(Salary, 1, 0)` تعني:

- العمود الذي سيتم إرجاع قيمته من الصيغ التالي → `Salary`.
- عدد الصيغ التي سيتم الرجوع إليها (صف واحد للأمام) → `1`.
- القيمة الافتراضية إذا لم يكن هناك صيغ تالي → `0`.

3. FIRST_VALUE() - جلب أول قيمة في المجموعة

❖ تُستخدم لاسترجاع أول قيمة في مجموعة بيانات مرتبة.

◆ مثال:

```
SELECT
    EmployeeID,
    Name,
    Salary,
    FIRST_VALUE(Salary) OVER(ORDER BY EmployeeID) AS FirstSalary
FROM Employees;
```

✓ هنا `FIRST_VALUE(Salary)` تعني:

- سيتم جلب أول راتب في القائمة وفق ترتيب `EmployeeID`.

4. LAST_VALUE() - جلب آخر قيمة في المجموعة

❖ تُستخدم لاسترجاع آخر قيمة في مجموعة بيانات مرتبة.

◆ مثال:

```
SELECT
    EmployeeID,
    Name,
    Salary,
    LAST_VALUE(Salary) OVER(ORDER BY EmployeeID ROWS BETWEEN UNBOUNDED
                           PRECEDING AND UNBOUNDED FOLLOWING) AS LastSalary
FROM Employees;
```

✓ هنا `LAST_VALUE(Salary)` سُترجع آخر راتب في الجدول.

◆ ولكن يجب تحديد النطاق باستخدام:

ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING

لضمان أن يتم جلب آخر صف في كامل الجدول وليس فقط في النطاق الحالي.

يعني من غير متكتب السطر دا هيجبلك آخر صف على حسب الترتيب اللي مرتب بيها مش علي حسب الجدول الأصلي

💡 مقارنة بين LAG g LEAD g FIRST_VALUE g LAST_VALUE

الدالة	الوظيفة
<code>LAG()</code>	استرجاع قيمة الصف السابق

<code>LEAD()</code>	استرجاع قيمة الصف التالي
<code>FIRST_VALUE()</code>	استرجاع أول قيمة في المجموعة
<code>LAST_VALUE()</code>	استرجاع آخر قيمة في المجموعة

⌚ متى تُستخدم هذه الدوال؟

مفیدان عند مقارنة القيم بين الصفوف المتباورة `LAG()` و `LEAD()`.

مفیدان عند الرغبة في معرفة أول أو آخر قيمة في مجموعة معينة `FIRST_VALUE()` و `LAST_VALUE()`.



What is a Table Scan?

A **Table Scan** happens when the database **searches through every row in a table** to find the required data. This is usually inefficient for large tables because the database has to read all the data, even if only a few rows match the query.

◆ Example of a Table Scan

Scenario:

You have a table called `Employees` with **1 million rows**, and you run this query:

```
SELECT * FROM Employees WHERE Salary = 5000;
```

If there is **no index** on the `Salary` column, the database has to **check every row** one by one to find the matches. This is a **table scan**.

◆ When Does a Table Scan Happen?

- ✓ **No Index on the column in the `WHERE` clause.**
- ✓ **Query needs to check most or all rows.**
- ✓ **Small tables (sometimes a full scan is faster than using an index).**

◆ How to Avoid Table Scans?

- ✓ **Use Indexes** on frequently searched columns.
- ✓ **Use SELECT only for necessary columns** instead of `SELECT *`.
- ✓ **Use WHERE conditions efficiently** to filter results early.

◆ Table Scan vs. Index Scan

Type	How It Works	Performance
Table Scan	Reads every row in the table	🔴 Slow for large tables
Index Scan	Reads the index instead of the full table	🟡 Faster but not always the best
Index Seek	Directly finds matching data using the index	✅ Fastest

📌 **Summary:** A **table scan** means searching **every row** in a table. It's slow for large tables but can be avoided with indexes. 🚀



what happen when you search with PK?

تلقياً، مما يتم استخدام **Primary Key (PK)**, يبحث في جدول يحتوي على **Clustered Index** عندما يتم استخدام **Table Scan**. يجعل البحث أسرع بكثير مقارنة بعمليات

1 ما نوع الـ Index المستخدم مع الـ Primary Key؟

- عليه، مما يعني أن البيانات **Clustered Index** على جدول، يتم تلقياً إنشاء **Primary Key** عند إنشاء بناءً على هذا المفتاح **(Physically Sorted)** في الجدول ترتيب مادياً.
- بدلاً من **Table Scan**, يتم استخدام **Index Seek** عندما تبحث عن قيمة باستخدام الـ **Primary Key**, مما يجعل البحث أسرع.

2 العلاقة بين البحث في الجداول والTREES

أكثر نوع لتمثيل الفهارس (**Indexes**) تستخدم قواعد البيانات **هياكل بيانات شجرية** **B-Tree** شائع من هذه الهياكل هو

- ◆ للعنور على البيانات بسرعة، بدلاً من **B-Tree** يتم استخدام شجرة **Primary Key**, عند البحث باستخدام فحص جميع الصفوف.
- ◆ كان البحث أسرع، لأنه يتطلب عدداً أقل من الخطوات للوصول، **(Balanced)** كلما كانت الشجرة متوازنة إلى البيانات.

3 ما نوع TREE المستخدمة في الفهارس؟

أكثر **B+ Tree** لأن **B+ Tree Index** وليس **Binary Tree** أو **AVL Tree**, مما يجعل البحث أسرع ويسهل التعامل مع كميات كبيرة من البيانات.

العادي؟ لماذا **B+ Tree** وليس **B-Tree**؟

1. **B+ Tree** يخزن البيانات في الأوراق فقط **Leaf Nodes**.
2. للأوراق المجاورة، مما يسهل تنفيذ **Pointers** تحتوي على مؤشرات **B+ Tree** كل عقدة في **Range Queries**.
3. أقل، مما يعني أن البحث أسرع مقارنة بهياكل أخرى مثل **Depth (Number of Levels)** **AVL Tree**.

4 كيف يتم البحث باستخدام **B+ Tree**؟

◆ على **EmployeeID** مع **PK** جدول **Employees** مثال: لديك

```
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
```

```

        Name VARCHAR(100),
        Salary INT
    );

```

فإن البيانات يتم تخزينها فعليًا وفقًا لهذا الترتيب، ويتم إنشاء **Primary Key** وهو `EmployeeID` بما أن **Clustered Index** يُستخدم **B+ Tree**.

◆ ماذا يحدث عند تنفيذ هذا الاستعلام؟

```
SELECT * FROM Employees WHERE EmployeeID = 10;
```

كما يلي يتم تنفيذ **Index Seek**:

1. في شجرة **B+ Tree** يبدأ البحث من **Root Node**.
2. للوصول إلى النطاق الذي يحتوي على القيمة المطلوبة **Internal Nodes** ينتقل إلى.
3. حيث يتم العثور على البيانات المخزنة فعليًا **Leaf Node** يصل إلى.
4. يتم إرجاع الصف المطلوب مباشرة دون الحاجة إلى فحص كل الصفوف.

📌 مقارنة بين Table Scan و Index Seek

النوع	كيف يعمل؟	الأداء
Table Scan	يبحث في كل صف داخل الجدول	بطيء جدًا
Index Scan	يبحث في كل صف داخل الفهرس	متوسط
Index Seek	ينتقل مباشرةً إلى البيانات باستخدام B+ Tree	أسرع



What is Difference between Clustered Index & Non-Clustered Index ?

تساعد في تحسين سرعة البحث واسترجاع البيانات. يوجد نوعان **Indexes** في قواعد البيانات، الفهارس أساسيات من الفهارس:

1. Clustered Index (الفهرس المترافق)

- هو الفهرس الأساسي الذي يحدد الترتيب الفعلي للبيانات داخل الجدول.
- يعنى أن **صفوف الجدول نفسها يتم تخزينها على القرص** بترتيب الفهرس.
- واحد فقط** لأن البيانات لا يمكن أن تكون مرتبة بأكثر **Clustered Index** يمكن أن يكون لكل جدول من طريقة واحدة.
- فإن قاعدة البيانات تصل إلى البيانات مباشرةً **Clustered Index**, عند البحث عن بيانات باستخدام دون الحاجة إلى مؤشر إضافي.

مثال عملي:

فسيتم تخزين الصفوف، لو كان لدينا جدول **employees** وفيه **Clustered Index** على **EmployeeID**, فعلى **EmployeeID** مرتبة حسب

- إذا طلبت البحث عن موظف معين برقم 5, فقاعدة البيانات ستصل مباشرةً إلى الصنف المخزن في الموضع الصحيح.

2. Non-Clustered Index (الفهرس غير المترافق)

- لا يغيّر ترتيب البيانات الفعلي داخل الجدول، بل ينشئ **هيكلًا منفصلًا** يحتوي على **المفتاح المفهرس** + **Pointers** إلى الموضع الفعلي للبيانات على القرص (مؤشرات).
- على نفس الجدول **Non-Clustered Index** يمكن أن يكون هناك أكثر من.
- يتم أولاً البحث داخل الفهرس، ثم استخدام **المؤشر** **Non-Clustered Index**, عند البحث باستخدام **Pointer** للوصول إلى الصنف الفعلي في الجدول.

مثال عملي:

عند البحث عن اسم معين، قاعدة البيانات ستبحث داخل الفهرس أولاً، ثم تستخدم **المؤشر** للوصول إلى الصنف المطلوب.

المقارنة بين Clustered g Non-Clustered Index

المعيار	Clustered Index	Non-Clustered Index
ترتيب البيانات	يخزن البيانات مرتبة فعليًا على القرص	لا يؤثر على ترتيب البيانات، بل يخزن فقط مؤشرات إلى الصفوف

عدد الفهارس لكل جدول	يمكن أن يكون واحد فقط	يمكن أن يكون أكثر من واحد
أداء البحث	أسرع عند البحث عن بيانات محددة باستخدام الفهرس الأساسي	يحتاج إلى خطوتين (البحث داخل الفهرس ثم الوصول إلى البيانات)
حجم الفهرس	يأخذ مساحة أقل لأن البيانات نفسها مفهرسة	يأخذ مساحة أكبر بسبب وجود المؤشرات الإضافية
أداء عمليات التحديث والإدراج	أبطأ عند التحديث أو الإدراج لأن البيانات يعاد ترتيبها	أسرع في التحديث والإدراج لأنه لا يؤثر على ترتيب البيانات

الخلاصة:

- يُخزن البيانات **بترتيب الفهرس نفسه**, سريع في البحث ولكنه يؤثر على الأداء عند تحرير البيانات.
 - يُخزن **المفتاح + المؤشر فقط**, مما يسمح بإنشاء **عدة فهارس** لكنه يتضيّف على خطوة إضافية عند البحث.
 - الأفضلية تعتمد على الاستخدام**
 - إذا كنت تبحث عن بيانات كثيرة بناءً على **مفتاح أساسي** → استخدم
 - إذا كنت بحاجة إلى البحث في أعمدة مختلفة بدون تغيير ترتيب البيانات → استخدم **Non-Clustered Index**.
- اختيار الفهرس المناسب يساعد في تحسين أداء قواعد البيانات بشكل كبير ✨



What are the types of indexes that created by default ?

PK ⇒ is a constrain that has a clustered index

Unique ⇒ is a Constrain That has a Nonclustered index



How Clustered Index Change his work ?

في الجدول عندما يكون هناك 1. Clustered Index

- بدلاً من **Clustered Index** بالإشارة إلى قيمة الـ **Non-Clustered Index** في هذه الحالة، يقوم الإشارة مباشرة إلى موقع البيانات في القرص.
- فإن **Clustered Index** لأن البيانات في الجدول مرتبة فعلياً وفقاً لـ **Non-Clustered Index** كمؤشر للوصول إلى الصف المطلوب **Clustered Index** يستخدم مفتاح الـ **Clustered Index**.
- يتم استرجاع البيانات بسرعة لأنه يحتوي على ترتيب البيانات، **Clustered Index** بعجرد الوصول إلى الفعلى.

مثال عملي:

لو كان لدينا جدول **الموظفين** يحتوي على:

- Clustered Index** على **EmployeeID**
- Non-Clustered Index** على **EmployeeName**

ثم يتم استخدام **EmployeeID** بإرجاع **EmployeeID** عند البحث عن موظف بالاسم، يقوم **Clustered Index** بتحديد الصف الفعلى.

في الجدول لا يوجد 2. Clustered Index (Heap Table)

- (**Heap**). هنا، الجدول يتم تخزينه بدون ترتيب محدد، أي أن البيانات موزعة عشوائياً في الصفحات.
- يحتوي على مؤشر مباشر إلى موقع البيانات الفعلى **Non-Clustered Index** في هذه الحالة، فإن **Non-Clustered Index** على القرص (Data Page).
- هذا يجعل البحث أبطأ مقارنةً بالحالة الأولى، لأن المؤشر يجب أن يشير إلى موقع عشوائي بدلاً من الاعتماد على ترتيب الفهرس المتجمع.

مثال عملي:

على **Non-Clustered Index** ولكن يحتوي على **Clustered Index** لو كان لدينا جدول **الموظفين** بدون **EmployeeName** فإن البحث عن موظف بالاسم سيحتاج إلى الوصول إلى المؤشر داخل **Non-Clustered Index**. ثم القفز إلى الموقع الفعلى للبيانات مباشرةً.

المقارنة بين الحالتين

الحالة	كيف يعمل Non-Clustered Index؟	الأداء
مع Clustered Index	ثم يتم، Clustered Index يشير إلى مفتاح استخدامة للوصول إلى البيانات الفعلية.	أسرع، لأن البيانات مرتبة وفقاً للفهرس الأساسي.
بدون Clustered Index (Heap Table)	يشير مباشرةً إلى موقع الصف في القرص الصلب.	أبطأ، لأن البيانات مخزنة بشكل غير منظم (Heap).

الخلاصة:

- يشير إلى مفتاحه وليس إلى الصف **Clustered Index**, فإن **Non-Clustered Index** إذا كان هناك مباشرة.
- يشير مباشرةً إلى موقع البيانات **Non-Clustered Index**, فإن **Clustered Index** إذا لم يكن هناك على الها رد، مما قد يجعل عمليات البحث أقل كفاءة.
- في معظم الجداول، حيث يُحسن من أداء الفهرس **Clustered Index** لهذا السبب، يفضل وجود الأخرى. 



What is Difference between SQL Server Profiler vs. SQL Server Tuning Advisor?

1. SQL Server Profiler

- أداة لمراقبة الأنشطة اللي بتحصل في قاعدة البيانات في الوقت الحقيقي.
- ومشاكل الأداء Deadlocks بتساعدك تتبع الاستعلامات اللي بتتنفذ، تسجيلات الدخول، الأخطاء، إلخ.
- مفيدة لو عايز تفهم ليه استعلام معين بطيء أو تشوّف العمليات اللي بتحصل في السيرفر.

مثال عملني:

- عن شان تشوّف إلا Profiler لو عندك استعلام بطيء ومش عارف السبب، ممكن تستخدم Query التي بتتنفذ ومدى استهلاكها للموارد.

2. SQL Server Tuning Advisor

- أو تحسين (Indexes) وتقترح حلول زي إضافة فهارس Workload أداة لتحسين الأداء بتحليل الاستعلامات.
- أو الاستعلامات اللي بتدخلها يدوياً عن شان SQL Profiler بتسخدم البيانات اللي تم تسجيلها في تقدم اقتراحات لتحسين الأداء.
- وإحصائيات Indexes، Partitions، من خلال اقتراح محشنة.

مثال عملني:

- ممكن يقترح عليك إضافة Tuning Advisor عندك جدول ضخم والاستعلامات عليه بطيئة؟ على عمود معين لتسرّع البحث Non-Clustered Index.

إيه الفرق بينهم؟

الأداة	وظيفتها	بتسخدمها في إيه؟
SQL Server Profiler	يترصد وتراقب كل اللي بيحصل في قاعدة البيانات لايف	لو عايز تتبع استعلامات بطيئة أو تشوّف المستخدمين بيعملوا إيه
SQL Server Tuning Advisor	تحليل الأداء وتقترح حلول لتحسينه	لو عندك استعلامات بطيئة وعايز تعرف إزاي تسرّعها



What is a System Databases ?

القاعدة	الوصف	الوظيفة الأساسية	ملاحظات هامة
master	قاعدة البيانات الرئيسية	تحتوي على كل المعلومات الحيوية للسيفر، مثل تفاصيل قواعد البيانات الأخرى، تسجيلات الدخول، الإعدادات، والأذونات.	مش SQL Server إذا تلفت لها Backup هيشتغل، ولازم تعمل باستمرار.
msdb	قاعدة إدارة المهام	تحتوي على بيانات SQL Server المجدولة، Jobs مثل إداره النسخ الاحتياطي، التنبهات، وخدمات البريد الإلكتروني.	تُستخدم في جدولة المهام، مثل إعادة بناء الفهارس تلقائياً.
model	القاعدة النموذجية	تُستخدم ك قالب افتراضي عند إنشاء أي قاعدة بيانات جديدة، وتأخذ منها الإعدادات الأساسية مثل الحجم الافتراضي وخيارات الاسترداد.	هينعكس model أي تعديل في على أي قاعدة بيانات جديدة يتم إنشاؤها بعد ذلك.
tempdb	قاعدة البيانات المؤقتة	تحتوي على الجداول المؤقتة، البيانات المؤقتة، وُتُستخدم في العمليات الداخلية مثل Sorting و Joins .	يتم إعادة إنشائها من الصفر مع فلات SQL Server، لا تحفظ بيانات دائمة. وتكون محفوظة على الرام او الذاكرة المؤقتة.

◆: الفرق الأساسي بين كل قاعدة بيانات

1. قاعدة البيانات الأساسية اللي بتحتوي على بيانات كل قواعد البيانات الأخرى = قاعدة البيانات الرئيسية.
2. قاعدة بيانات إدارة المهام والمجدولة، زي تشغيل المهام تلقائياً وإدارة النسخ الاحتياطي = قاعدة بيانات إدارة المهام والمجدولة.
3. قالب اللي بيتم نسخه عند إنشاء قاعدة بيانات جديدة = قالب.
4. في العمليات الداخلية والجداول SQL Server قاعدة بيانات مؤقتة بيسخدمها = قاعدة بيانات المؤقتة.

✓ باختصار:

- مش هيشتغل master → SQL Server لو فقدت!
- المهام المجدولة والنسخ الاحتياطي مش هتشتغل msdb → لو فقدت.
- مش هيأثر مباشرة، بس القواعد الجديدة مش هتكون مضبوطة صح → model لو فقدت.
- بس ممكن يسبب بطء في العمليات المؤقتة tempdb → لو فقدت.



How can you create 10 Deference databases that contain same table Ex (Employee) by create table with one query ?

to do that you can use system database Model and create in it a table Employee and create the 10 database ,then all databases will contain the same table Employee



What is a Local Table in SQL Server?

A **local table** generally refers to a **temporary table** that is only accessible within the session (connection) that created it.

◆ Types of Local Tables:

1. Local Temporary Table (`#TempTable`)
2. Table Variable (`@TableVariable`)

1 Local Temporary Table (`#TempTable`)

- Created using `#` before the table name.
- Exists **only for the session** that created it.
- Automatically deleted when the session ends.
- لما تيجي تعمل New query in MSS كدا بتعمل جديدة

Example:

```
CREATE TABLE #TempEmployees (
    ID INT PRIMARY KEY,
    Name NVARCHAR(100)
);

INSERT INTO #TempEmployees VALUES (1, 'Ahmed'), (2, 'Sara');

SELECT * FROM #TempEmployees; -- Only visible in this session
```

When the session closes, `#TempEmployees` is deleted automatically.

2 Table Variable (`@TableVariable`)

- Defined using `@` before the variable name.
- **Only exists within the batch or stored procedure** where it's declared.
- Cannot have indexes (except primary keys).

Example:

```
DECLARE @TempEmployees TABLE (
    ID INT PRIMARY KEY,
    Name NVARCHAR(100)
);
```

```
INSERT INTO @TempEmployees VALUES (1, 'Ali'), (2, 'Mona');
```

```
SELECT * FROM @TempEmployees; -- Only available in this batch
```

✓ `@TempEmployees` disappears after execution ends.

◆ Key Differences:

Feature	#TempTable	@TableVariable
Scope	Visible in the session	Limited to the batch
Stored in	tempdb	In memory (or tempdb if large)
Supports Indexes?	Yes	Only primary key
Used in Transactions?	Yes	No (rollback doesn't affect it)
Performance	Good for large data	Better for small data

🛠 When to Use Which?

- Use `#TempTable` when you need indexes, large data, or cross-batch usage.
- Use `@TableVariable` for small data and when working inside stored procedures.

🚀 Both are useful for performance optimization in SQL Server!



What is a Global Table in SQL Server?

A **global table** usually refers to a **Global Temporary Table** in SQL Server. It is a temporary table that is accessible by multiple sessions (connections) at the same time.

◆ Global Temporary Table (`##TempTable`)

- Created using `##` (double hash) before the table name.
- Accessible by all sessions** (unlike local temporary tables, which are session-specific).
- Remains available **until the last session that references it is closed**.
- DB زی فکرة ال کل و هيinthل يعدل وفالآخر بعد متخلص نحفظ في ال meeting

Example:

```
CREATE TABLE ##GlobalTemp (
    ID INT PRIMARY KEY,
    Name NVARCHAR(100)
);

INSERT INTO ##GlobalTemp VALUES (1, 'Ahmed'), (2, 'Sara');

SELECT * FROM ##GlobalTemp; -- Accessible in any session
```

If another session queries `##GlobalTemp`, it will see the same data!

◆ Key Differences Between Local and Global Temporary Tables

Feature	<code>#TempTable</code> (Local)	<code>##TempTable</code> (Global)
Scope	Only visible in the session that created it	Visible to all sessions
Stored in	<code>tempdb</code>	<code>tempdb</code>
Deletion	Deleted when the session ends	Deleted when the last session using it closes
Use Case	Session-specific operations	Sharing data across multiple connections

❖ When to Use Global Temporary Tables?

- When multiple sessions need access to temporary data.
- When different users or procedures need to share intermediate results.
- Be careful! If one session **deletes or modifies** the table, it affects all other sessions using it.



Use wisely to avoid unexpected data modifications!



Compare between ROLLUP, CUBE, and GROUPING SETS ?

These are advanced **GROUP BY** extensions used for generating subtotals, grand totals, and multi-level aggregations.

1 What is ROLLUP ?

ROLLUP creates hierarchical totals by progressively aggregating from **detailed level** → **subtotal** → **grand total**.

Example (Single Column ROLLUP)

```
SELECT Department, SUM(Salary) AS TotalSalary  
FROM Employees  
GROUP BY ROLLUP(Department);
```

✓ Results:

Department	TotalSalary
IT	10000
HR	7000
NULL (Grand Total)	17000

Example (Multi-Column ROLLUP)

```
SELECT Department, JobTitle, SUM(Salary) AS TotalSalary  
FROM Employees  
GROUP BY ROLLUP(Department, JobTitle);
```

✓ Results:

Department	JobTitle	TotalSalary
IT	Developer	5000
IT	Manager	5000
IT	NULL	10000
HR	Analyst	4000
HR	Manager	3000
HR	NULL	7000
NULL	NULL	17000

2 What is CUBE ?

CUBE creates all possible **combinations of aggregations** (cross-tabulation).

Example (Multi-Column CUBE)

```
SELECT Department, JobTitle, SUM(Salary) AS TotalSalary  
FROM Employees  
GROUP BY CUBE(Department, JobTitle);
```

✓ Results:

Department	JobTitle	TotalSalary
IT	Developer	5000
IT	Manager	5000
IT	NULL	10000
HR	Analyst	4000
HR	Manager	3000
HR	NULL	7000
NULL	Developer	5000
NULL	Manager	8000
NULL	Analyst	4000
NULL	NULL	17000

🚀 Difference from ROLLUP:

- **ROLLUP** aggregates in a hierarchy (`Department → JobTitle → Total`).
- **CUBE** aggregates across **all possible combinations**.

3 What is **GROUPING SETS** ?

GROUPING SETS allow custom groupings without enforcing a hierarchy.

Example (Custom Grouping Sets)

```
SELECT Department, JobTitle, SUM(Salary) AS TotalSalary  
FROM Employees  
GROUP BY GROUPING SETS (  
    (Department, JobTitle), -- Normal Grouping  
    (Department),        -- Aggregate by Department  
    (JobTitle),          -- Aggregate by JobTitle  
    ()                  -- Grand Total  
);
```

✓ Results:

Department	JobTitle	TotalSalary
IT	Developer	5000
IT	Manager	5000
HR	Analyst	4000
HR	Manager	3000
IT	NULL	10000
HR	NULL	7000
NULL	Developer	5000
NULL	Manager	8000
NULL	Analyst	4000
NULL	NULL	17000

🚀 More flexible than ROLLUP & CUBE because it allows custom grouping combinations.

◆ 5 Key Use Cases

Case	Method	Purpose
1 Department-wise Salary with Grand Total	<code>ROLLUP(Department)</code>	Gives department-level and overall total salary.
2 All Combinations of Department & JobTitle	<code>CUBE(Department, JobTitle)</code>	Generates cross-tabulated results.
3 Custom Grouping of Department & JobTitle	<code>GROUPING SETS ((Department, JobTitle), (Department), (JobTitle), ())</code>	Provides maximum flexibility in aggregations.
4 Multi-Level Aggregation	<code>ROLLUP(Region, Country, City)</code>	Aggregates from City → Country → Region → Grand Total.
5 Category & Product-wise Aggregation	<code>CUBE(Category, Product)</code>	Provides all subtotal combinations for better reporting.

🚀 Use `ROLLUP` for hierarchical aggregation, `CUBE` for full combinations, and `GROUPING SETS` for full control! 🚀



What is deference between Pivot & Unpivot ?

📌 **PIVOT** - تحويل الصفوف إلى أعمدة

بالشكل ده مثلاً: عندك جدول **sales**:

ProductID	SalesmanName	Quantity
1	Ahmed	100
2	Ahmed	150
1	Khalid	200
2	Khalid	250

🎯 عايزين نحول البيانات بحيث يكون **Ahmed** و **Khalid** أعمدة:

```
SELECT *
FROM sales
PIVOT (
    SUM(Quantity)
    FOR SalesmanName IN ([Ahmed], [Khalid])
) AS PVT;
```

📌 **PIVOT** : النتيجة بعد

ProductID	Ahmed	Khalid
1	100	200
2	150	250

✓ كده البيانات بقت معروضة في شكل **أفقي** (**أعمدة**) بدل ما كانت في **صفوف**.

📌 **UNPIVOT** - تحويل الأعمدة إلى صفات

عايزين نرجع البيانات لشكلها الأصلي

```
SELECT ProductID, SalesmanName, Quantity
FROM SalesPivotTable
UNPIVOT (
    Quantity
    FOR SalesmanName IN (Ahmed, Khalid)
) AS Unpvt;
```

📌 **النتيجة بعد UNPIVOT :**

ProductID	SalesmanName	Quantity
1	Ahmed	100
1	Khalid	200
2	Ahmed	150
2	Khalid	250

✓ رجّعنا البيانات لشكلها الطبيعي (صفوف بدل أعمدة).

🔥 خلاصة سريعة

- ◆ PIVOT → يحول الصفوف إلى أعمدة لعرض البيانات بطريقة مجمعة.
- ◆ UNPIVOT → يرجع الأعمدة إلى صفوف لجعل البيانات أكثر مرؤنة.



📌 What is View ?

هو كأنك بتعمل "جدول وهمي" بيعتمد على استعلام معين، وبقدر View في SQL Server عادي بدون ما تخزن البيانات فعلياً Table تستخدموه كأنه.

◆ استخدامات لـ View:

- تسهيل التعامل مع البيانات المعقّدة
- تقليل التعقيد في الاستعلامات الكبيرة
- تحسين الأمان بإخفاء بعض الأعمدة من الجدول الأصلي
- تحسين الأداء في بعض الحالات

🔥 أنواع لـ View في SQL Server:

1 Standard View (العادي)

عادي بدون أي تعقيبات `SELECT` بيعتمد على Views، هو أبسط نوع من لـ View.
لا يخزن بيانات فعلية، بل ينفذ الاستعلام كل مرة يتم استدعاوه
يمكن استخدامه لقراءة البيانات لكنه لا يدعم التعديلات في بعض الحالات.

مثال:

```
CREATE VIEW EmployeeView AS
SELECT EmployeeID, Name, Department
FROM Employees;
```

لـ View هنا لا يخزن البيانات، بل يعتمد على البيانات الأصلية في `Employees`.

2 Indexed View (المفهرس)

ما يخزن البيانات فعلياً باستخدام فهرس View يختلف عن لـ Standard View، يجعله أسرع في القراءة.

- ليمנע التعديلات على الجداول الأساسية يحتوي على View يجب أن يكون الـ SCHEMABINDING
- وبعدها يمكن إضافة **Clustered Index**، يتم إنشاؤه باستخدام **Non-Clustered Indexes**.

مثال:

```
CREATE VIEW EmployeeSalaryView
WITH SCHEMABINDING
AS
SELECT EmployeeID, SUM(Salary) AS TotalSalary
FROM dbo.Employees
```

GROUP BY EmployeeID;

```
-- إنشاء View "薪水 مرتبة" و هذا يجعل (I) ملحوظ
CREATE UNIQUE CLUSTERED INDEX IX_EmployeeSalaryView
ON EmployeeSalaryView (EmployeeID);
```

أفضل أداء عند تنفيذ الاستعلامات، لكنه يحتاج مساحة تخزين إضافية Indexed View: ميزة لا

3 Partitioned View (المجذأ)

في قواعد بيانات (Partitioned Tables) يستخدم عندما تكون البيانات مقسمة على أكثر من جدول مختلفة أو نفس القاعدة.

- يسعى ب التقسيم للبيانات لزيادة الأداء وتوزيع العمل على أكثر من خادم.
 - أو موزعاً (Local) أو موزعاً (Distributed).**

 في نفس السيرفر (Local Partitioned View) مثال على:

```
CREATE VIEW AllEmployees AS  
SELECT * FROM Employees_Part1  
UNION ALL  
SELECT * FROM Employees_Part2;
```

💡 يقلل الضغط على قاعدة البيانات لأنه يسمح بتقسيم البيانات على عدة Partitioned View ميزة لا جداول أو سيرفرات.

مقارنة بين الأنواع الثلاثة:

النوع	التخزين	الأداء	يقبل التعديلات؟	الاستخدام الأساسي
Standard View	لا يخزن البيانات 	بطيء أحياناً 	<input checked="" type="checkbox"/> نعم (شروط)	تبسيط الاستعلامات
Indexed View	يخزن البيانات 	أسرع بكثير 	<input checked="" type="checkbox"/> نعم	تحسين الأداء في الاستعلامات
Partitioned View	لا يخزن البيانات 	جيد جدًا مع التقسيم 	 لا يدعم التعديلات مباشرةً	توزيع البيانات وتحسين الأداء

الخلاصة

- ◆ **Standard View:** مجرد استعلام مخزن لتحسين سهولة القراءة.
 - ◆ **Indexed View:** أسرع لكنه يحتاج فهرسة وت تخزين إضافي.
 - ◆ **Partitioned View:** يقسم البيانات على جداول مختلفة لتحسين الأداء.
 - 💡 اختيار النوع المناسب يعتمد على حجم البيانات والأداء المطلوب.

✓ **Indexed vs Partitioned Views**



How can Haden table from users and appear it to admins only ?

✓ الخطوات:

1. **جديد للمسؤولين فقط Schema إنشاء:**

```
CREATE SCHEMA AdminSchema AUTHORIZATION dbo;
```

2. **الجديد Schema نقل الجدول إلى الـ:**

```
ALTER SCHEMA AdminSchema TRANSFER dbo.Employees;
```

3. **إلغاء صلاحيات جميع المستخدمين العاديين على هذا الـ Schema:**

```
DENY SELECT ON SCHEMA::AdminSchema TO Public;
```

💡 **الآن الجدول مخفي عن جميع المستخدمين، ولن يظهر إلا للمسؤول 🔥**



How can encryption view to avoid hacking ?

Hacker can show your code of view by use `SP_helptext` + view Name

to avoid this encryption view when you created it by use

```
CREATE VIEW EmployeeView  
With encryption  
AS  
SELECT EmployeeID, Name, Department  
FROM Employees;
```

but after this no one can show script of view

and when you generation a script file wit encryption views it will be get Error



◆ في SQL Server مع الـ **VIEW** مع الـ (**INSERT** , **UPDATE** , **DELETE**) شروط استخدام أوامر DML

ولكن هناك **شروط** يجب، **VIEW** عند التعامل مع تتوفرها حتى تكون التعديلات على البيانات ممكنة.

✓ مع الشروط الأساسية لاستخدام **VIEW**

يجب أن يتحقق ، **VIEW** وينفذ (**INSERT** , **UPDATE** , **DELETE**) قابلًا للتحديث لكي يكون الشروط التالية:

1 على جدول واحد فقط **VIEW** يجب أن يعتمد

- إذا كان يحتوي على أكثر من جدول باستخدام **JOIN** لا يمكن التعديل على **INSTEAD OF TRIGGER** .

2 عدم استخدام **DISTINCT**

- لأنه يجعل الصفوف غير مميزة **VIEW** في **INSERT** , **UPDATE** , **DELETE** وجود

3 عدم استخدام **GROUP BY** أو **HAVING**

- هذه العبارات تغير هيكل البيانات، مما يجعل التعديل غير ممكن.

4 عدم استخدام الدوال التجميعية (**SUM** , **AVG** , **COUNT**)

- هذه الدوال تؤثر على البيانات، مما يمنع تعديلها مباشرة.

5 على جميع الأعمدة المطلوبة **VIEW** يجب أن يحتوي **NOT NULL Columns**

- فلن يعمل ، **VIEW** غير موجودة في الـ (**NOT NULL**) إذا كان الجدول يحتوي على أعمدة إجبارية **INSERT** .

6 أو **UNION** أو **UNION ALL** عدم استخدام

- فلا يمكن تعديل البيانات ، **VIEW** أو **UNION** يحتوي على **UNION ALL** إذا كان

7 على "مفتاح أساسي" أو "مفتاح فريد" **VIEW** يجب أن يحتوي

- لضمان إمكانية تحديد كل صف بشكل فريد عند التعديل

◆ أمثلة عملية

✓ مثال **VIEW** قابل للتحديث (Updatable View)

```
CREATE VIEW EmployeeView AS
SELECT EmployeeID, Name, Salary
FROM Employees;
```

- لأنه يعتمد على **جدول واحد فقط**، ولا يمكن استخدامه مع **VIEW** هذا يحتوي على أي دوال تجميعية أو **GROUP BY** .

إدخال بيانات:

```
INSERT INTO EmployeeView (EmployeeID, Name, Salary)
VALUES (101, 'Ahmed', 5000);
```

تحرير بيانات:

```
UPDATE EmployeeView
SET Salary = 6000
WHERE EmployeeID = 101;
```

حذف بيانات:

```
DELETE FROM EmployeeView
WHERE EmployeeID = 101;
```

✗ VIEW غير قابل للتحديث (Non-Updatable View)

✗ VIEW يحتوي على JOIN

```
CREATE VIEW EmployeeDepartmentView AS
SELECT e.EmployeeID, e.Name, d.DepartmentName
FROM Employees e
JOIN Departments d ON e.DepartmentID = d.DepartmentID;
```

يعتمد على أكثر من جدول **المشكلة**: لا يمكن إدخال أو تحرير أو حذف بيانات، لأن

✗ VIEW يحتوي على GROUP BY

```
CREATE VIEW DepartmentSalary AS
SELECT DepartmentID, AVG(Salary) AS AvgSalary
FROM Employees
GROUP BY DepartmentID;
```

المشكلة: لا يمكن تعديل لأنه يعتمد على **AVG(Salary)** **GROUP BY**.

🔥 قابل للتحديث VIEW الحلول لجعل

1 استخدام INSTEAD OF TRIGGER

- يحتوي على على يمكن إنشاء **Trigger** ليس مع بتنفيذ **INSERT**, **UPDATE**, **DELETE** **VIEW** **JOIN**.

2 استخدام WITH CHECK OPTION لمنع التعديلات غير الصحيحة*

```
CREATE VIEW EmployeeView AS
SELECT EmployeeID, Name, Salary
FROM Employees
WHERE Salary > 3000
WITH CHECK OPTION;
```

- من إدخال بيانات لا تتوافق مع شرط WHERE .

❸ الخلاصة

- ✓ يمكنك استخدام INSERT , UPDATE , DELETE مع VIEW بشرط:
 - ❖ دوال تجتمعية على VIEW ، JOIN ، GROUP BY ، DISTINCT ، UNION ، HAVING .
 - ❖ على جدول واحد فقط VIEW أن يعتمد على جدول آخر (NOT NULL).
 - ❖ يعتمد على أكثر من جدول كدل بديل لتحديث VIEW يمكن استخدام INSTEAD OF TRIGGER .



why we use **WITH CHECK OPTION** ?

❖ فائدتها؟

في الـ **WHERE** يتمتع إدخال أو تعديل بيانات تخالف شروط

❖ بدونها؟

لأنها لا تتحقق الشرط **VIEW** ممكّن تدخل أو تعديل بيانات لكنها تخفي من

❖ مثال عملي:

```
CREATE VIEW HighSalary AS  
SELECT Name, Salary FROM Employees  
WHERE Salary > 5000  
WITH CHECK OPTION;
```

- مسعوه: إدخال راتب **6000** لأنه يحقق الشرط.

- مرفوض: إدخال أو تعديل راتب **4000** لأنه أقل من **5000**.

❖ الخلاصة: تحافظ على البيانات المتوافقة مع شروط **VIEW**!





How can **MERGE** ?

في نفس الجملة بناءً على مقارنة **MERGE** هي **كلمة مفتاحية** تتيح لك إجراء **INSERT** و **UPDATE** و **DELETE** بين جداولين.

معنني بستخدمها لما اعوز احدث مثلا جدول بجدول اخر

⌚ متى نستخدم **MERGE** ؟

- بيانات جديدة إذا لم تكن موجودة (**INSERT**) عند الحاجة إلى إضافة.
- البيانات لو كانت موجودة (**UPDATE**) عند الحاجة إلى تحديث.
- بيانات غير متطابقة مع الجدول المصدر (**DELETE**) عند الحاجة إلى حذف.

📌 مثال عملي

عندنا جدولين:

- ◆ يحتوي على البيانات الحالية **جدول Target**.
- ◆ يحتوي على البيانات الجديدة التي نريد دمجها **جدول Source**.

```
MERGE INTO Employees AS Target
USING NewEmployees AS Source
ON Target.EmployeeID = Source.EmployeeID
```

```
-- ✓ تحديث البيانات لو الموظف موجود
WHEN MATCHED THEN
    UPDATE SET Target.Salary = Source.Salary

-- ✓ إدخال بيانات جديدة لو الموظف غير موجود
WHEN NOT MATCHED THEN
    INSERT (EmployeeID, Name, Salary)
    VALUES (Source.EmployeeID, Source.Name, Source.Salary)

-- ✓ حذف الموظفين غير الموجودين في المصدر
WHEN NOT MATCHED BY SOURCE THEN
    DELETE;
```

شرح الكود ببساطة

- 1 يحدث راتبه → () لو الموظف موجود **MATCHED**.
- 2 يتم إدخاله → () لو الموظف غير موجود **NOT MATCHED**.
- 3 يحذف → () لو الموظف في **Target** وليس في **Source** **NOT MATCHED BY SOURCE**.



الخلاصة:

`MERGE` بتسهيل التحديث والإدخال والحذف في عملية واحدة بدل ما تكتب كل وحدة لوحدة.



What is a Query Execution Steps ?

1 Parsing (التحليل)

☒ الخاص بالاستعلام (Syntax) يتم فحص بناء الجملة.

☑ إذا كان هناك خطأ نحوبي، يتم رفض الاستعلام وإرجاع خطأ Syntax Error.

◆ مثال:

```
SELECT * FROM students; -- SELECT يجب أن تكون خطأ نحوبي
```

2 Binding - التحقق من البيانات الوصفية (Metadata Validation)

☒ التتحقق مما إذا كانت الجداول والأعمدة المشار إليها موجودة في قاعدة البيانات أم لا.

☑ إذا كان أي جدول أو عمود غير موجود، يتم إرجاع خطأ Object Not Found.

◆ مثال:

```
SELECT name FROM employees; -- employees خطأ إذا كان جدول غير موجود
```

3 Optimization (تحسين الاستعلام)

يقوم محرك قاعدة البيانات بإنشاء خط تنفيذ مختلف للاستعلام ويختر الأفضل منها بناءً على عدة عوامل مثل الفهارس (Indexes) والإحصائيات (Statistics).

☑ الهدف هو تنفيذ الاستعلام بأقل تكلفة زمنية ممكنة.

◆ مثال:

- على عمود معين، فسيتم استخدامه بدلاً من البحث في كل الصفوف (Index) إذا كان هناك فهرس.
- بين جداول كبيرة، سيتم اختيار أفضل ترتيب للانضمام (Join Order).

4 Query Tree (إنشاء شجرة الاستعلام)

☒ تم تحويل الاستعلام إلى شجرة استعلام منظمة.

☑ الشجرة تحدد ترتيب العمليات مثل الفرز، التصفية، والتجميع.

◆ مثال (للاستعلام التالي):

```
SELECT name FROM students WHERE age > 18;
```

▼ شجرة الاستعلام (Query Tree):

1. **FROM students** تحديد مصدر البيانات.

2. WHERE age > 18 ✓ تصفية الصفوف بناءً على الشرط.

3. SELECT name ✓ اختيار العمود المطلوب.

👉 هذه الشجرة تساعد في إنشاء خطة التنفيذ (Execution Plan).

5 Execution Plan Generation (إنشاء خطة التنفيذ)

بناءً على شجرة الاستعلام، يتم إنشاء خطة التنفيذ الفعلية، والتي تحدد كيفية جلب البيانات.

قاعدة البيانات تختار أفضل طريقة للتنفيذ مثل ✓:

◆ بدلاً من فحص الجدول بالكامل (Index Seek) استخدام فهرس (Table Scan).

◆ بناءً على حجم البيانات JOIN أو Hash Join تنفيذ بطريقة.

6 Execution (تنفيذ الاستعلام)

يتم تنفيذ الخطة المحددة، واسترجاع النتائج من الذاكرة أو القرص.

✓ لتسريع الاستعلامات (Cache) إذا كانت البيانات مطلوبة عدة مرات، قد يتم تخزينها في الذاكرة المستقبلية.

7 Return Result (إرجاع النتيجة إلى التطبيق أو المستخدم)

إلى المستخدم أو التطبيق الذي نفذ الاستعلام (ResultSet) يتم إرسال مجموعة النتائج.

◆ مثال:

```
SELECT * FROM students WHERE id = 4;
```

سيتم إرسال صف واحد فقط إلى التطبيق بدلاً من إرسال كل بيانات الطلاب، مما يقلل من استهلاك (Network Traffic).

مثال عملي لتوضيح كل المراحل

استعلام SQL:

```
SELECT name FROM students WHERE age > 18;
```

كيف يتم تنفيذه؟

1 Parsing: التتحقق من صحة الكلمات (SELECT, FROM, WHERE).

2 Binding: التأكد من وجود students و age .

3 Optimization: تحديد ما إذا كان هناك فهرس على age .

4 Query Tree: (FROM → WHERE → SELECT).

5 Execution Plan: اختيار أفضل طريقة لجلب البيانات.

6 Execution: تنفيذ الخطة وجلب البيانات.

7

Return Result: إرسال النتائج إلى التطبيق أو المستخدم.



What is Stored Procedure ?

- ◆ **Stored Procedure** هي مجموعة من أوامر SQL يمكنها دخول قاعدة البيانات، واستدعاءها وتنفيذها عند الحاجة.
- ◆ في كل مرة، يمكن استدعاء الـ SQL بدلاً من إرسال استعلام باسمها فقط.
- ◆ **لجلب بيانات الطلاب حسب العمر مثلاً: إنشاء**

```
CREATE PROCEDURE GetStudentsByAge
    @Age INT
AS
BEGIN
    SELECT * FROM students WHERE age > @Age;
END;
```

لاستدعائها:

```
EXEC GetStudentsByAge 18;
```

في الأداء، الشبكة، والأمان فوائد الـ **Stored Procedures**

1) تحسين الأداء (Performance)

- ✓ مرة واحدة عند إنشائه، مما يجعله أسرع في (Query Optimization) يتم ترجمة وتحسين الاستعلام التنفيذ مقارنةً بإرسال استعلام جديد في كل مرة.
- ✓ مما يقلل وقت المعالجة عند (Execution Plan) في الذاكرة (Cache)، يتم تخزين خطة التنفيذ استدعائها عدة مرات.

◆ **مقارنة الأداء:**

طريقة التنفيذ	هل يتم ترجمة الاستعلام؟	الأداء
() استعلام عادي	يتم ترجمته في كل مرة	أبطأ
Stored Procedure	يتم ترجمته مرة واحدة	أسرع

2) تقليل استهلاك الشبكة (Network Traffic)

- ✓ فقط مع **Stored Procedure** كامل في كل مرة، يتم إرسال اسم الـ SQL بدلاً من إرسال استعلام المعاملات المطلوبة.
- ✓ يقلل كمية البيانات المرسلة بين التطبيق وقاعدة البيانات، مما يحسن الأداء خاصة عند التعامل مع كمية كبيرة من البيانات.

◆ مثال:

● الطريقة العادية:

```
SELECT * FROM students WHERE age > 18;
```

● استخدام Stored Procedure:

```
EXEC GetStudentsByAge 18;
```

يتم إرسال كل نص الاستعلام إلى السيرفر، أما في الـ إرسال اسم الإجراء فقط مع المعاملات، مما يقلل استهلاك الشبكة.

3 تحسين الأمان (Security)

لأن الاستعلامات مخزنة مسبقاً ولا يتم تنفيذها بناءً على مدخلات المستخدم **SQL Injection** يمكن منع المباشرة.

ب بحيث يمكن للمستخدم تحديد الصلاحيات **Stored Procedure** فقط دون الوصول المباشر إلى الجداول.

◆ مثال لمنع SQL Injection:

● الطريقة العادية (عرضة للهجموم):

```
string query = "SELECT * FROM students WHERE name = '" + userInput + "'";
```

! فسيقوم بإرجاع جميع الصفوف إذا كان `userInput = ' OR 1=1 --`

● استخدام Stored Procedure (آمن):

```
EXEC GetStudentsByName @StudentName;
```

حتى لو أدخل المستخدم بيانات ضارة، لن يتم تنفيذ استعلام خطير.

4 تسهيل الصيانة وإعادة الاستخدام (Maintainability & Reusability)

واحدة، مما يسهل **Stored Procedure** بدلاً من تكرار نفس الكود في كل مكان، يتم تخزينه في التعديل والصيانة.

بدون الحاجة إلى تعديل **Stored Procedure** إذا احتجت إلى تغيير طريقة الاستعلام، يمكنك تحديث لا التطبيق!

● الخلاصة: متى نستخدم الـ **Stored Procedures**؟

إذا كنت تريد تحسين الأداء

إذا كنت تريد تقليل استهلاك الشبكة

إذا كنت تريد أماناً أعلى ومنع **SQL Injection**



إذا كنت تريده كوداً أسهل في الصيانة وإعادة الاستخدام ✓



Stored Procedures !أداة قوية لتحسين أداء وأمان التطبيقات التي تتعامل مع قواعد البيانات 🔥



what is a types of Stored Procedures?

◆ يوجد 3 أنواع رئيسية من الـ Stored Procedures:

1) الإجراءات المخزنة المضمنة (Built-in Stored Procedures)

2) الإجراءات المخزنة المعرفة من قبل المستخدم (User-defined Stored Procedures)

3) المحفزات (Triggers)

1) الإجراءات المخزنة المضمنة (Built-in Stored Procedures)

✓ وتببدأ دائمًا بـ "sp_" أو "sys." في SQL Server، وهذه إجراءات مخزنة جاهزة داخل

✓ تُستخدم لأغراض إدارية مثل إدارة قواعد البيانات، تشغيل النسخ الاحتياطي، فحص الأخطاء، أو جلب معلومات عن الجداول.

◆ أمثلة:

◆ عرض جميع الجداول في قاعدة البيانات:

```
EXEC sp_tables;
```

◆ الحصول على بنية جدول معين:

```
EXEC sp_help 'students';
```

◆ إعادة تسمية جدول أو عمود:

```
EXEC sp_rename 'old_table_name', 'new_table_name';
```

◆ الموجودة في قاعدة البيانات (Schemas) معرفة المخططات:

```
EXEC sp_databases;
```

📌 متى نستخدمها؟

- عند الحاجة إلى إدارة قاعدة البيانات بدون كتابة استعلامات معقدة.
- عند البحث عن معلومات حول الجداول والبيانات المخزنة.

2) الإجراءات المخزنة المعرفة من قبل المستخدم (User-defined Stored Procedures)

✓ هذه الإجراءات يتم إنشاؤها بواسطة المستخدم لتنفيذ عمليات مخصصة.

✓ يمكنها قبول **Parameters** (Result Sets).

◆ لجلب بيانات الطلاب حسب العمر Stored Procedure مثال: إنشاء وإدارة

```
CREATE PROCEDURE GetStudentsByAge
    @Age INT
AS
BEGIN
    SELECT * FROM students WHERE age > @Age;
END;
```

📌 لاستدعائها:

```
EXEC GetStudentsByAge 18;
```

📌 متى نستخدمها؟

- عندما يكون لدينا عمليات متكررة مثل الإدراجه، التحديث، البحث، إلخ.
- لتحسين الأداء وتقليل استهلاك الشبكة.
- لتوفير أمان أعلى ومنع SQL Injection.

③ 3 المحفزات (Triggers) 🔥

هو نوع خاص من الإجراءات المخزنة يتم تشغيله تلقائياً عند حدوث حدث معين داخل أو قاعدة البيانات.

✓ أو حذف، (DELETE)، (UPDATE)، (INSERT) لا يمكن استدعاؤه يدوياً، بل يتم تشغيله عند إدراجه (DELETE)، (UPDATE)، (INSERT).

◆ لمنع حذف البيانات من جدول الطلاب Trigger مثال: إنشاء

```
CREATE TRIGGER PreventDeleteStudents
ON students
INSTEAD OF DELETE
AS
BEGIN
    PRINT 'لا يمكن حذف بيانات الطلاب !'
    ROLLBACK TRANSACTION;
END;
```

📌 إذا حاول شخص تنفيذ:

```
DELETE FROM students WHERE id = 1;
```

● لن يتم الحذف، وسيظهر التنبيه

📌 متى نستخدمها؟

- لـ**لـ العمـاـيـة الـبـيـانـات** منـ الحـذـف أوـ التـعـدـيل غـير المـقصـود.
- لـ**لـتـنـفـيـذ عـمـلـيـات تـلـقـائـيـة** عـنـدـ حدـوثـ تـغـيـيرـاتـ عـلـىـ الـبـيـانـات.
- فـيـ نـظـامـ مـعـيـنـ (Audit Log) عـنـدـ الحاجـةـ إـلـىـ حـفـظـ سـجـلـ الـعـمـلـيـات.

● مـقـارـنـة بـيـنـ الـأـنـوـاعـ الـثـلـاثـةـ

النوع	طـرـيـقـةـ التـشـغـيلـ	الاستـخـدـامـ الرـئـيـسـيـ
Built-in Stored Procedures	(يتم استدعاؤها يدوياً) <code>EXEC sp_xxx</code>	إدارة قاعدة البيانات والعمليات الإدارية
User-defined Stored Procedures	(يتم استدعاؤها يدوياً) <code>EXEC procedure_name</code>	تنفيذ عمليات مخصصة، تحسين الأداء والأمان
Triggers	يتم تشغيلها تلقائياً عند حدوث حدث (<code>INSERT</code> , <code>UPDATE</code> , <code>DELETE</code>)	فرض القواعد، تسجيل العمليات، منع التعديلات غير المصرح بها

● الـخـلاـصـةـ: متـىـ تـسـتـخـدـمـ كـلـ نـوـعـ؟

- ✓ **Built-in Procedures** → عندما تحتاج إلى استعلامات جاهزة لإدارة قاعدة البيانات.
- ✓ **User-defined Procedures** → عند تنفيذ استعلامات مخصصة لتحسين الأداء وتقليل استهلاك الشبكة.
- ✓ **Triggers** → عندما تحتاج إلى تشغيل عمليات تلقائية عند تغيير البيانات، مثل منع حذف سجلات معينة.
- 💡 ذكاء يساعد في تحسين الأداء، الأمان، وتقليل استهلاك الموارد **Stored Procedures** استخدام لا في قاعدة البيانات! 🔥



when we use Return in store procedure?

لإرجاع قيمة رقمية (RETURN) داخل الإجراء المخزن يمكن استخدام SQL Server . وعادةً ما تُستخدم للإشارة إلى حالة التنفيذ أو رمز خطأ (INT) واحدة فقط.

◆ القواعد الأساسية لاستخدام RETURN في Stored Procedure

- أو بيانات معقدة (RETURN) إرجاع قيم نصية لا يمكن لـ VARCHAR .
- يتم استخدامه غالباً لإرجاع كود نجاح أو فشل (Status Code).
- يتم إنهاء الإجراء فوراً ولا يتم تنفيذ أي كود بعده ، عند استخدام RETURN .
- إذا لم يتم تحديدها هي 0 (تعني نجاح التنفيذ) القيمة الافتراضية لـ RETURN .

● مثال 1: استخدام RETURN لإرجاع كود نجاح أو فشل

```
CREATE PROCEDURE CheckDoctorExists
    @DoctorId INT
AS
BEGIN
    IF EXISTS (SELECT 1 FROM Doctors WHERE DoctorId = @DoctorId)
        RETURN 1; -- ✓ الطبيب موجود
    ELSE
        RETURN 0; -- ✗ الطبيب غير موجود
END;
```

استدعاء الإجراء المخزن واستقبال القيمة

```
DECLARE @Result INT;
EXEC @Result = CheckDoctorExists 5;

IF @Result = 1
    PRINT '✓.الطبيب موجود في النظام';
ELSE
    PRINT '✗.الطبيب غير موجود في النظام';
```

● مثال 2: استخدام RETURN مع كود خطأ

```
CREATE PROCEDURE DeleteDoctor
    @DoctorId INT
```

```

AS
BEGIN
-- التحقق من وجود الطبيب أو لاً
IF NOT EXISTS (SELECT 1 FROM Doctors WHERE DoctorId = @DoctorId)
BEGIN
    PRINT ' خطأ: الطبيب غير موجود';
    RETURN -1; -- خطأ إرجاع كود
END

-- حذف الطبيب
DELETE FROM Doctors WHERE DoctorId = @DoctorId;

PRINT ' تم حذف الطبيب بنجاح';
RETURN 0; -- نجاح
END;

```

استدعاء الإجراء المخزن ومعرفة النتيجة

```

DECLARE @Status INT;
EXEC @Status = DeleteDoctor 3;

IF @Status = 0
    PRINT ' تم تنفيذ الحذف بنجاح';
ELSE
    PRINT ' حدث خطأ أثناء التنفيذ';

```

🔴 متى لا تستخدم RETURN ؟

✖️ إذا كنت تريد إرجاع أكثر من قيمة، فاستخدم **SELECT** أو **TABLE RETURN** بدلاً من **RETURN**.
 ✖️ لا يمكنه إرجاع بيانات معقدة مثل النصوص أو الجداول.

🔵 مقارنة بين RETURN و OUTPUT

الميزة	RETURN	OUTPUT Parameter
نوع البيانات	فقط INT	أي نوع بيانات (INT, VARCHAR, DATETIME, etc.)
عدد القيم المعادة	قيمة واحدة فقط	يمكن إرجاع أكثر من قيمة
الاستخدام الأساسي	(إرجاع كود حالة) Status Code	إرجاع بيانات مفصلة

◆ مثال سريع على **OUTPUT Parameter** :

```

CREATE PROCEDURE GetDoctorName
    @DoctorId INT,

```

```
@DoctorName NVARCHAR(100) OUTPUT  
AS  
BEGIN  
    SELECT @DoctorName = DoctorName FROM Doctors WHERE DoctorId = @DoctorId;  
END;
```

استدعاء الإجراء مع **OUTPUT**

```
DECLARE @Name NVARCHAR(100);  
EXEC GetDoctorName 2, @Name OUTPUT;  
PRINT @Name;
```

الخلاصة

- ✓ أو رمز خطأ (**RETURN Status Code**) فقط لإرجاع كود حالة.
- ✓ إذا كنت تريد إرجاع بيانات نصية أو أكثر من قيمة استخدم **OUTPUT Parameter**.
- ✓ إذا كنت تريد إرجاع بيانات متعددة الصيغ استخدم **SELECT**.

 الاختبار يعتمد على متطلبات التطبيق الخاصة بك!



How can call by reference with **Stored Procedure** ?

لإرجاع قيم متعددة مثل النصوص أو الأرقام أو التواريخ، يمكنك استخدام **OUTPUT** مع **Stored Procedure** واحد فقط (**INT**) الذي يمكنه إرجاع قيمة عدديّة على عكس **RETURN**.

قواعد استخدام **OUTPUT** في **Stored Procedure**

- لإرجاع البيانات بعد تنفيذ الإجراء المخزن (**Parameters**) في المعاملات **OUTPUT** يستخدم.
- لإرجاع بيانات متعددة يمكن استخدام **أكثر من OUTPUT Parameter**.
- بل يمكن متابعة تنفيذ الأوامر بعده ، إلى إنهاء الإجراء المخزن مثل **RETURN** لا يؤدي.
- عند استدعاء الإجراء المخزن لاستقبال القيمة **OUTPUT** يجب تحديد.

مثال 1: إرجاع اسم الطبيب باستخدام **OUTPUT**

الإجراء المخزن

```
CREATE PROCEDURE GetDoctorInfo
    @DoctorId INT,
    @DoctorName NVARCHAR(100) OUTPUT
AS
BEGIN
    SELECT @DoctorName = DoctorName FROM Doctors WHERE DoctorId = @DoctorId;
END;
```

استدعاء الإجراء المخزن واستقبال القيمة

```
DECLARE @Name NVARCHAR(100);
EXEC GetDoctorInfo 2, @Name OUTPUT;
PRINT 'اسم الطبيب: ' + @Name;
```

مثال 2: إرجاع بيانات متعددة باستخدام **OUTPUT**

الإجراء المخزن يعيد معلومات الطبيب كاملة

```

CREATE PROCEDURE GetDoctorDetails
    @DoctorId INT,
    @DoctorName NVARCHAR(100) OUTPUT,
    @Specialization NVARCHAR(50) OUTPUT
AS
BEGIN
    SELECT @DoctorName = DoctorName, @Specialization = Specialization
    FROM Doctors WHERE DoctorId = @DoctorId;
END;

```

استدعاء الإجراء المخزن واستقبال القيم

```

DECLARE @Name NVARCHAR(100), @Spec NVARCHAR(50);
EXEC GetDoctorDetails 3, @Name OUTPUT, @Spec OUTPUT;
PRINT 'اسم الطبيب ' + @Name;
PRINT 'التخصص ' + @Spec;

```

مثال 3: استخدام **OUTPUT** مع عملية **INSERT**

بعد إدراج بيانات جديدة (**ID**) لإرجاع المعرف يمكن استخدام **OUTPUT**.

الإجراء المخزن لإضافة طبيب وإرجاع **DoctorId**

```

CREATE PROCEDURE AddDoctor
    @DoctorName NVARCHAR(100),
    @Specialization NVARCHAR(50),
    @NewDoctorId INT OUTPUT
AS
BEGIN
    INSERT INTO Doctors (DoctorName, Specialization)
    VALUES (@DoctorName, @Specialization);

    -- إرجاع آخر معرف تم إدخاله
    SET @NewDoctorId = SCOPE_IDENTITY();
END;

```

استدعاء الإجراء المخزن واستقبال الجديد **DoctorId**

```

DECLARE @DoctorId INT;
EXEC AddDoctor 'عيون', 'د. أحمد', @DoctorId OUTPUT;
PRINT 'تمت إضافة الطبيب الجديد بمعرف ' + CAST(@DoctorId AS NVARCHAR);

```

الفرق بين **OUTPUT** و **RETURN**

الميزة	OUTPUT	RETURN
عدد القيم المعادة	يمكن إرجاع أكثر من قيمة	قيمة واحدة فقط (INT)
نوع البيانات	أي نوع بيانات (VARCHAR, INT, DATETIME, etc.)	INT فقط
الاستخدام الأساسي	إرجاع بيانات مثل الأسماء أو التفاصيل	إرجاع كود حالة (Status Code)

الخلاصة

 **OUTPUT** يستخدم لإرجاع بيانات معقدة أو متعددة القيم.

 **RETURN** (**Status Code**) يستخدم لإرجاع كود حالة.

 المضاف حديثاً للحصول على **OUTPUT** مع **INSERT ID** يمكن استخدام.

 عندما تحتاج إلى إرجاع بيانات متعددة من **Stored Procedure** !



📌 What is a Dynamic Query?

A **dynamic query** is an SQL query that is **built and executed at runtime** using `EXEC` or `sp_executesql`.

It is useful when the query structure depends on **user input** or **dynamic conditions**.

✅ Executing a Dynamic Query with `EXEC`

```
DECLARE @TableName NVARCHAR(100) = 'Doctors';
DECLARE @Query NVARCHAR(MAX);

SET @Query = 'SELECT * FROM ' + @TableName;
EXEC(@Query);
```

◆ Here, the table name is stored in the `@TableName` variable, and the `SELECT` statement is built dynamically.

✅ Executing a Dynamic Query with `sp_executesql`

```
DECLARE @Query NVARCHAR(MAX);
DECLARE @DoctorId INT = 3;

SET @Query = 'SELECT * FROM Doctors WHERE DoctorId = @Id';
EXEC sp_executesql @Query, N'@Id INT', @DoctorId;
```

◆ Here, the `@Id` parameter is passed securely to **prevent SQL Injection**.

🎯 When to Use Dynamic Queries?

✓ When the **table or column name is unknown** at the time of writing the query.

✓ When applying **dynamic filtering** based on user input.

✓ When dynamically selecting **columns or tables**.

🚀 **Note:** It's better to use `sp_executesql` instead of `EXEC` because it is **more secure** against SQL Injection.

◆ Dynamic Query (Simple Explanation)

📌 What is a Dynamic Query?

A **dynamic query** is an SQL query that is **built and executed at runtime** using `EXEC` or `sp_executesql`.

It is useful when the query structure depends on **user input** or **dynamic conditions**.

✓ Executing a Dynamic Query with `EXEC`

```
DECLARE @TableName NVARCHAR(100) = 'Doctors';
DECLARE @Query NVARCHAR(MAX);

SET @Query = 'SELECT * FROM ' + @TableName;
EXEC(@Query);
```

◆ Here, the table name is stored in the `@TableName` variable, and the `SELECT` statement is built dynamically.

✓ Executing a Dynamic Query with `sp_executesql`

```
DECLARE @Query NVARCHAR(MAX);
DECLARE @DoctorId INT = 3;

SET @Query = 'SELECT * FROM Doctors WHERE DoctorId = @Id';
EXEC sp_executesql @Query, N'@Id INT', @DoctorId;
```

◆ Here, the `@Id` parameter is passed securely to **prevent SQL Injection**.

🎯 When to Use Dynamic Queries?

- ✓ When the **table or column name is unknown** at the time of writing the query.
 - ✓ When applying **dynamic filtering** based on user input.
 - ✓ When dynamically selecting **columns or tables**.
- 🚀 **Note:** It's better to use `sp_executesql` instead of `EXEC` because it is **more secure** against SQL Injection.



What is a trigger ?

A **trigger** in databases is a special type of stored procedure that automatically executes in response to certain events on a specified table or view. Triggers are commonly used to enforce business rules, maintain data integrity, and automate actions in a database.

Types of Triggers

1. AFTER Triggers (also called FOR Triggers)

- Executes **after** an `INSERT`, `UPDATE`, or `DELETE` operation.
- Commonly used for logging changes or enforcing constraints.

2. INSTEAD OF Triggers

- Executes **instead of** an `INSERT`, `UPDATE`, or `DELETE` operation.
- Used when you want to modify the behavior of an operation, such as redirecting changes to another table.

3. DDL Triggers (Data Definition Language)

- Fires in response to schema changes like `CREATE TABLE`, `ALTER TABLE`, or `DROP TABLE`.

4. Logon Triggers

- Executes when a user logs into the database, often used for security and monitoring.

Example of an AFTER INSERT Trigger (SQL Server)

```
CREATE TRIGGER trg_AfterInsert
ON Patients
AFTER INSERT
AS
BEGIN
    INSERT INTO AuditLog (TableName, Action, TimeStamp)
    VALUES ('Patients', 'INSERT', GETDATE())
END;
```

This trigger logs any new patient insertion into an `AuditLog` table.



How can use After and instead of and how created a read-only table ?

1. AFTER Trigger

This trigger will log any insertion into the `Patients` table.

```
CREATE TABLE AuditLog (
    Id INT IDENTITY(1,1) PRIMARY KEY,
    TableName NVARCHAR(50),
    Action NVARCHAR(50),
    TimeStamp DATETIME DEFAULT GETDATE()
);

CREATE TRIGGER trg_AfterInsertPatients
ON Patients
AFTER INSERT
AS
BEGIN
    INSERT INTO AuditLog (TableName, Action, TimeStamp)
    VALUES ('Patients', 'INSERT', GETDATE());
END;
```

◆ **What this does:** Every time a new patient is added to the `Patients` table, an entry is recorded in the `AuditLog`.

2. INSTEAD OF Trigger

This trigger prevents deletion from the `Doctors` table and logs the attempt.

```
CREATE TRIGGER trg_InsteadOfDeleteDoctors
ON Doctors
INSTEAD OF DELETE
AS
BEGIN
    INSERT INTO AuditLog (TableName, Action, TimeStamp)
    VALUES ('Doctors', 'DELETE ATTEMPT', GETDATE());

    PRINT 'Deletion is not allowed on Doctors table!';
END;
```

- ◆ **What this does:** If someone tries to delete a doctor, the action is blocked, and a log entry is created instead.

3. Read-Only Table Using a Trigger

We will make a table **read-only** by preventing `INSERT`, `UPDATE`, and `DELETE` operations.

Step 1: Create the Read-Only Table

```
CREATE TABLE ReadOnlyData (
    Id INT PRIMARY KEY,
    Info NVARCHAR(100)
);
```

Step 2: Create the Trigger

```
CREATE TRIGGER trg_PreventModification_ReadOnlyData
ON ReadOnlyData
INSTEAD OF INSERT, UPDATE, DELETE
AS
BEGIN
    PRINT 'This table is read-only. Modifications are not allowed.';
END;
```

- ◆ **What this does:** Any attempt to modify `ReadOnlyData` will be blocked.

Summary

- `AFTER` Trigger: Logs inserts in `Patients`.
- `INSTEAD OF` Trigger: Prevents deletion in `Doctors`.
- **Read-Only Table:** Blocks modifications in `ReadOnlyData`.



How can control in triggers ?

1. Disable a Trigger

To temporarily disable a trigger without deleting it:

```
DISABLE TRIGGER trg_AfterInsertPatients ON Patients;
```

This will prevent the trigger `trg_AfterInsertPatients` from executing when an insert operation happens on the `Patients` table.

2. Enable a Trigger

To re-enable the trigger after disabling it:

```
ENABLE TRIGGER trg_AfterInsertPatients ON Patients;
```

Now, the trigger will execute again when an insert occurs.

3. Disable All Triggers on a Table

```
DISABLE TRIGGER ALL ON Patients;
```

This disables **all** triggers associated with the `Patients` table.

4. Enable All Triggers on a Table

```
ENABLE TRIGGER ALL ON Patients;
```

This enables **all** triggers on the `Patients` table.

When to Use This?

- If you want to **temporarily stop** a trigger while making bulk inserts/updates.
- If a trigger is causing **unexpected behavior**, and you want to debug it.



What happen after created a trigger ?

by default two tables create :

inserted and deleted tables

inserted ⇒ contain the new data that insert

deleted ⇒ contain the last data or deleted data

```
create trigger t1
on Sales.OrderDetails
after update
as
select * from INSERTED
select * from deleted

update Sales.OrderDetails
set Quantity =100
where id =1
```

	Id	OrderId	ProductId	UnitPrice	Quantity
1	1	1	5	14.99	100

	Id	OrderId	ProductId	UnitPrice	Quantity
1	1	1	5	14.99	2



How can not delete in Friday only ?

```
create trigger t3
on Sales.OrderDetails
instead of delete
as
IF (DATENAME(WEEKDAY, GETDATE()) != 'Friday')
begin
delete from Sales.OrderDetails where ProductId in (select ProductId from deleted)
end
--here after try to delete by default data store in table deleted
--سواء حذف او حاول يحذف--
```

III



How can use trigger to secure ⇒ know who deleted ?

```
create trigger t1
on sales
instead of update
as
if UPDATE(ProductID)
begin
    declare @oldid int
    declare @newid int
    select @oldid = ProductID from deleted
    select @newid = ProductID from inserted
    insert into history values
        (SUSER_NAME(),GETDATE(),@oldid,@newid)
end
```

```
disable trigger onreadonly on sales
disable trigger t1 on sales
```

```
drop trigger onreadonly
drop trigger t1
```

```
select * from sales
```

```
update sales
set ProductID =100
where SalesmanName ='yoyo' --not updated her
```

```
select * from history --here data of who deleted appear
```



How can use Keyword Output ?

it is a runtime trigger

استخدام **OUTPUT** مع **UPDATE** :

إذا كنت تريد رؤية القيم قبل وبعد التحديث، يمكنك استخدام

```
UPDATE sales  
SET ProductID = 100  
OUTPUT deleted.ProductID AS OldProductID, inserted.ProductID AS NewProductID  
WHERE SalesmanName = 'yowdfyo';
```

◆ **deleted** : تحتوي على البيانات قبل التحديث.

◆ **inserted** : تحتوي على البيانات بعد التحديث.

استخدام **OUTPUT** مع **INSERT** :

إذا كنت تريد إرجاع القيم التي تم إدخالها حديثاً:

```
INSERT INTO sales (SalesmanName, ProductID)  
OUTPUT inserted.*  
VALUES ('John Doe', 200);
```

استخدام **OUTPUT** مع **DELETE** :

لمعرفة السجلات المحذوفة:

```
DELETE FROM sales  
OUTPUT deleted.*  
WHERE SalesmanName = 'yowdfyo';
```

◆ **فائدة OUTPUT** :

✓ تسمح لك بمراقبة التغييرات بسهولة.

✓ تستخدم أحياناً لحفظ التعديلات في سجل التدقيق (Audit Table).



How can Use XML for Database Conversion (MSSQL → Oracle)

XML can be used as an **intermediate format** to transfer data between different databases like **SQL Server and Oracle**. Here's how it helps:

1. Export Data from Source DB (MSSQL)

- Extract data from SQL Server in **XML format**.
- This ensures that data structure and relationships are preserved.

2. Transfer the XML File

- The generated XML file is moved to the target database system (Oracle).
- This can be done manually or via an automated pipeline.

3. Import Data into Target DB (Oracle)

- Oracle reads the XML file and **parses** it into its tables.
- XML ensures that the data remains **consistent and structured**.

Benefits of Using XML:

- ✓ **Universal Format** – XML can be understood by any database.
- ✓ **Preserves Relationships** – Foreign keys and nested data remain intact.
- ✓ **Scalable** – Suitable for both small and large data migrations.



What a deference ways to use xml in DB Conversion ?

1. FOR XML

FOR XML is used in **SQL Server** to format query results as XML.

Types of **FOR XML** modes:

1. **FOR XML RAW (Renamed as FOR XML SIMPLE)**

- Converts each row into an XML **element**, with column values as **attributes**.

Example Query:

```
SELECT SalesmanName, ProductID  
FROM Sales  
FOR XML RAW ('Transaction'); -- Renamed to 'Transaction' instead of 'row'
```

Output:

```
<Transaction SalesmanName="John" ProductID="101"/>  
<Transaction SalesmanName="Alice" ProductID="102"/>
```

What it does:

- Each row becomes an XML **element** named `<Transaction>`.
- Column values are stored as **attributes**.

2. **FOR XML ROOT (Renamed as FOR XML MAIN)**

- Wraps the entire XML result in a **single root element**.

Example Query:

```
SELECT SalesmanName, ProductID  
FROM Sales  
FOR XML RAW ('Transaction'), ROOT ('SalesData'); -- Renamed 'ROOT' to 'SalesData'
```

Output:

```
<SalesData>
  <Transaction SalesmanName="John" ProductID="101"/>
  <Transaction SalesmanName="Alice" ProductID="102"/>
</SalesData>
```

 **What it does:**

- Adds a **root** element `<SalesData>` around the entire XML structure.

3. FOR XML ELEMENTS (Renamed as FOR XML NODES)

- Converts column values into **nested elements** instead of attributes.

Example Query:

```
SELECT SalesmanName, ProductID
FROM Sales
FOR XML RAW ('Transaction'), ELEMENTS; -- Renamed 'ELEMENTS' to 'NODES'
```

Output:

```
<Transaction>
  <SalesmanName>John</SalesmanName>
  <ProductID>101</ProductID>
</Transaction>
<Transaction>
  <SalesmanName>Alice</SalesmanName>
  <ProductID>102</ProductID>
</Transaction>
```

 **What it does:**

- Instead of storing column values as **attributes**, they are stored as **child elements**.

Final Combined Query:

```
SELECT SalesmanName, ProductID
FROM Sales
FOR XML RAW ('Transaction'), ROOT ('SalesData'), ELEMENTS;
```

Final Output:

```

<SalesData>
  <Transaction>
    <SalesmanName>John</SalesmanName>
    <ProductID>101</ProductID>
  </Transaction>
  <Transaction>
    <SalesmanName>Alice</SalesmanName>
    <ProductID>102</ProductID>
  </Transaction>
</SalesData>

```

Key Differences:

SQL Option	Renamed	Effect
FOR XML RAW	FOR XML SIMPLE	Converts rows into elements with attributes
FOR XML ROOT	FOR XML MAIN	Wraps XML in a root element
FOR XML ELEMENTS	FOR XML NODES	Converts attributes into child elements

2. XML AUTO

it working with parent and child ⇒ join

SQL Query:

```

SELECT
  Customers.CustomerID, Customers.CustomerName,
  Orders.OrderID, Orders.OrderDate
FROM Customers
JOIN Orders ON Customers.CustomerID = Orders.CustomerID
FOR XML AUTO;

```

Output (XML Format):

```

<Customers CustomerID="1" CustomerName="John">
  <Orders OrderID="101" OrderDate="2024-02-27"/>
  <Orders OrderID="102" OrderDate="2024-02-28"/>
</Customers>
<Customers CustomerID="2" CustomerName="Alice">

```

```
<Orders OrderID="103" OrderDate="2024-02-29"/>  
</Customers>
```

Explanation:

- **FOR XML AUTO** automatically creates XML based on table relationships.
- The **JOIN** ensures `Orders` are nested inside `Customers`.

1. **FOR XML EXPLICIT**

- Gives full control over the XML structure.
- Requires manually specifying column levels.
- Example (complex query needed):

```
SELECT 1 AS Tag, NULL AS Parent, SalesmanName AS [Sales!1!SalesmanName], ProductID AS [Sales!1!ProductID]  
FROM Sales  
FOR XML EXPLICIT;
```

- Output:

```
<Sales>  
  <SalesmanName>John</SalesmanName>  
  <ProductID>101</ProductID>  
</Sales>
```

2. **FOR XML PATH**

- Best for creating **custom XML structures**.
- Example:

```
select  
  ProductID "@prId",  
  SalesmanName "SIName/@SN",  
  Quantity "SIName/Qy"  
  
from sales  
for xml path('salser')
```

- Output:

```

<salser prId="1">
  <SIName SN="ahmed">
    <Qy>10</Qy>
  </SIName>
</salser>
<salser prId="1">
  <SIName SN="khalid">
    <Qy>20</Qy>
  </SIName>
</salser>

```

2. OPENXML

OPENXML is used to **read XML data into SQL Server tables**.

Example:

1. Declare and Parse XML

```

DECLARE @xmlData XML =
'<SalesData>
  <salser prId="1">
    <SIName SN="ahmed">
      <Qy>10</Qy>
    </SIName>
  </salser>
  <salser prId="1">
    <SIName SN="khalid">
      <Qy>20</Qy>
    </SIName>
  </salser>
</SalesData>';

```

```

declare @n int;
exec sp_xml_preparedocument @n output,@xmlData

```

```

SELECT *
FROM OPENXML(@n, '/SalesData/salser')
WITH (
  prId int '@prId',
  SIName nvarchar(20) 'SIName/@SN',
  Qy int 'SIName/Qy'

```

```
);
```

Exec sp_xml_removedocument @n --for remove xml tree bu pinter of it

```
1 ahmed 10  
1 khalid 20
```

- 📌 Use **FOR XML** for exporting data to XML.
- 📌 Use **OPENXML** for importing XML into tables.



How can get a null in xml ?

Here's a simple SQL query to get **NULL** in XML:

```
SELECT Name, Age  
FROM Person  
FOR XML PATH('Person'), ELEMENTS XSINIL;
```

Explanation:

- **FOR XML PATH('Person')** → Converts data into XML format.
- **ELEMENTS XSINIL** → Ensures **NULL** values appear as **<Age xsi:nil="true" />** instead of being removed.



What is a Cursor ?

يستخدم للتعامل مع البيانات صفًّا بصف، وهو مفيد عندما تحتاج إلى تنفيذ عمليات متتابعة على كل صف بدلاً من التعامل مع البيانات دفعة واحدة.

- **Declare Cursor:** Define a cursor that holds the result of the SQL query.
- **Declare Variables:** Declare variables to store column values from the cursor.
- **Open Cursor:** Open the cursor to begin processing.
- **Fetch Row:** Retrieve the first row and assign values to variables.
- **Check Fetch Status (@fetch_status):**
 - 0 → Row fetched successfully → **Process the data**
 - 1 → No more rows to fetch
 - 2 → Fetch error
- **Processing:** Perform required operations on the fetched row.
- **Loop Back:** Fetch the next row and continue processing until no more rows exist.
- **Close Cursor:** Release the cursor after processing.
- **Deallocate Cursor:** Free up memory used by the cursor.



give me a code of looping to find all names in list

```
declare c2 cursor
for select SalesmanName
from sales
where SalesmanName is not null
for read only
declare @names nvarchar(1000),@name nvarchar(20)
open c2
fetch c2 into @name
while @@FETCH_STATUS=0
begin
set @names = concat(@names,'-',@name)
fetch c2 into @name
end
select @names
close c2
deallocate c2
```



give me a code of looping to update tables in specific case ?

```
declare c cursor
for select UnitPrice
from Products
for update
declare @Unt decimal(2)
open c
fetch c into @Unt
while @@FETCH_STATUS = 0
begin
if @Unt < 20.00
    update Products
    set UnitPrice =@Unt + 10.00
    where current of c
else
    update Products
    set UnitPrice =@Unt + 5.00
    where current of c
fetch c into @Unt
end
close c
deallocate c

select UnitPrice
from Products
```



Question: When is using a **CURSOR** beneficial compared to executing multiple queries?

✓ Answer: A **CURSOR** is useful when you need to **fetch data once from disk** and then apply **different updates to each row** sequentially, reducing repeated I/O operations. However, it is **slower than traditional queries**, so using **UPDATE** with **CASE** is preferred whenever possible. 🚀

ليس جيداً لتحسين الأداء في معظم الحالات. بل هو أبطأ من الاستعلامات التقليدية لأنه، نعم **CURSOR** يعالج كل صف على حدة بدلاً من تنفيذ العملية دفعة واحدة.

💡 خياراً سيئاً؟ متى يكون **CURSOR**؟

- عند التعامل مع **عدد كبير من السجلات**, لأنه يستهلك المزيد من الذاكرة ويبطئ التنفيذ.
- معاً يكون **أو**, واحد باستخدام SQL عندما يمكن استبداله باستعلام **JOIN** ، **UPDATE** ، **CASE** مع **CTE**.
- أسرع وأكثر كفاءة.

✓ متى يمكن أن يكون مفيداً؟

- إذا كنت بحاجة إلى **تنفيذ عمليات معقدة على كل سجل بشكل منفصل** ولا يمكن تنفيذها بسهولة باستعلام واحد.
- عندما تحتاج إلى **جلب البيانات مرة واحدة فقط** وتنفيذ عمليات متعددة عليها في الذاكرة، مما يقلل من عمليات القراءة من القرص.

◆ الخلاصة:

- ليس خياراً مثالياً لتحسين الأداء في معظم السيناريوهات **CURSOR**.
- استخدمه فقط إذا لم يكن هناك بديل أكثر كفاءة في SQL. 🚀



What are the Types of Backup ?

Backup Type	What It Backs Up	Restoration Process	File Type Used
Full Backup	all data in the database	restore full backup	.mdf + .ldf
Differential Backup	changes since the last full backup	restore full backup then differential backup	.mdf only
Transaction Log Backup	changes since the last backup	restore full backup then differential backup then transaction logs	.ldf only

one way to backup data in specific time is by **Transaction Log**

لأن في ملف الـ ldf يبقى موجود أوقات الكويريز

can apply backup with weather or code like :

```
backup database databasename  
to disk = location
```



How can solve Identity gap ?

by control in insert_identity

```
set identity_insert Employees on --to add in specific identity you want  
set identity_insert Employees off
```

```
select IDENT_CURRENT('Employees')--to get a last identity for specific table  
select @@IDENTITY --for last query
```



How can Bulk Insert ?

```
bulk insert Employees  
from yourfile  
WITH (  
FIELDTERMINATOR = ',',  
ROWTERMINATOR = '\n',
```



What is a Snapshot?

هو نسخة للقراءة فقط من قاعدة البيانات في لحظة زمنية معينة. تُستخدم لمراقبة البيانات في تلك اللحظة دون التأثر بالتغييرات التي تحدث بعد ذلك في قاعدة البيانات الأصلية.

مثال عملي في SQL Server

لنفترض أن لدينا جدولًا للطلاب يحتوي على بيانات كالتالي:

ID	الاسم	العمر
1	أحمد	22
2	إيمان	21
3	عمر	25
4	سالي	22

📌 **في الساعة 11 صباحاً يوم 30/11/2000، قمنا بإنشاء Snapshot باسم StudentSnapshot**

```
CREATE DATABASE StudentSnapshot
ON
(
    NAME = MyDatabase,
    FILENAME = 'C:\snapshots\StudentSnapshot.ss'
)
AS SNAPSHOT OF MyDatabase;
```

✓ **تبقي البيانات كما هي حتى لو تغيرت القاعدة الأصلية، Snapshot بعد إنشاء لا.**

ماذا يحدث عند التعديل في الجدول الأصلي؟

إذا قمنا بتحديث بعض البيانات في الجدول الأصلي:

```
UPDATE students
SET age = 28
WHERE name = 'سالي';
```

🔴 في قاعدة البيانات الأصلية، سيتم تعديل عمر سالي إلى 28.

✓ **ستظل البيانات كما كانت عند لحظة إنشاء (عمرها يبقى 22) لكن في لا.**

متى نستخدم لا Snapshot؟

الاحتفاظ بنسخة من البيانات في وقت محدد لمراجعة التغييرات لاحقاً.

تقليل الضغط على قاعدة البيانات عند الحاجة إلى تقارير عن البيانات القديمة.

تحليل البيانات في وقت معين دون التأثر بالتحديثات الحالية.

To restore DB from Snapshot

```
RESTORE DATABASE MyDatabase  
FROM DATABASE_SNAPSHOT = 'StudentSnapshot';
```



What is a SQL CLR (Common Language Runtime) in SQL Server?

SQL CLR allows you to write **Stored Procedures, Functions, Triggers, and Aggregates** using **C# or VB.NET** instead of T-SQL in **SQL Server**.

✓ Why Use SQL CLR?

- When **complex calculations** are needed.
- To **improve performance** for **heavy loops** or **advanced logic**.
- When working with **text processing, encryption, or .NET libraries**.

⚙️ Enable SQL CLR in SQL Server

Before using **SQL CLR**, you need to enable it in **SQL Server**:

```
sp_configure 'clr enabled', 1;
RECONFIGURE;
```

🛠️ Example: Creating a SQL CLR Function in C#

1 Create a .NET Library

- Open **Visual Studio**, create a **Class Library (.NET Framework)**
- Add references: **System.Data** & **Microsoft.SqlServer.Server**
- Create **MyClrFunctions.cs** and add:

```
using System;
using System.Data.SqlTypes;
using Microsoft.SqlServer.Server;

public class MyClrFunctions
{
    [SqlFunction]
    public static SqlString ReverseString(SqlString input)
    {
        if (input.IsNull)
            return SqlString.Null;
```

```
        char[] chars = input.Value.ToCharArray();
        Array.Reverse(chars);
        return new string(chars);
    }
}
```

2 Deploy and Test in SQL Server

1. **Copy the DLL file to SQL Server.**
2. **Register the assembly** in SQL Server:

```
CREATE ASSEMBLY MyClrAssembly
FROM 'C:\Path\To\Your\Dll\MyClrFunctions.dll'
WITH PERMISSION_SET = SAFE;
```

1. **Create the function** in SQL Server:

```
CREATE FUNCTION dbo.ReverseString(@input NVARCHAR(MAX))
RETURNS NVARCHAR(MAX)
AS EXTERNAL NAME MyClrAssembly.MyClrFunctions.ReverseString;
```

1. **Test the function:**

```
SELECT dbo.ReverseString('Hello SQL CLR');
```

Output: RLC QLS olleH

🎯 Important Notes

- ◆ **SQL CLR is powerful** but not always faster than T-SQL.
- ◆ Use **TRUSTWORTHY mode** if using `PERMISSION_SET = UNSAFE`.
- ◆ **Disable SQL CLR** with:

```
sp_configure 'clr enabled', 0;
RECONFIGURE;
```

🎯 Summary

- ✓ **SQL CLR allows C#/.NET functions inside SQL Server.**
- ✓ Useful for **complex logic and performance improvements**.
- ✓ Needs **to be enabled and deployed correctly**.

