**Insert + Join**

* EX1

All students who **live in Alexandria** (St\_Address = 'Alex')

And **are not already enrolled** in course 200

Will be **added** to course 200

With a **grade of 90**

**insert into Stud\_Course (Crs\_Id, St\_Id, Grade)**

**select 200, S.St\_Id, 90**

**from Student S left join Stud\_Course SC**

**on SC.Crs\_Id = 200 and SC.St\_Id = S.St\_Id**

**where S.St\_Address = 'Alex' and SC.St\_Id is null**

Selects all **students from Alexandria** (St\_Address = 'Alex') from the Student table.

Uses a LEFT JOIN to check if each student **is not already enrolled** in course 200.

The condition SC.St\_Id IS NULL ensures that only **students who are not yet in course 200** are selected.

Inserts those students into the Stud\_Course table, registering them in **course 200 with a fixed grade of 90**

* EX2 Insert into Stud\_Course (Crs\_Id, St\_Id, Grade)

select 1100, S.St\_Id, S.Grade

from Stud\_Course S

left join Stud\_Course SC

on SC.Crs\_Id = 1100 and SC.St\_Id = S.St\_Id

where S.Crs\_Id = 200 and SC.St\_Id IS NULL;

It selects **all students who are currently enrolled in course 200** from the Stud\_Course table.

It uses a LEFT JOIN to check if those students **are already registered** in course 1100.

The condition Existing.St\_Id IS NULL filters out any student who **is already enrolled** in course 1100.

The remaining students are inserted into the Stud\_Course table for course 1100 **with the same grades** they had in course 200

|  |  |
| --- | --- |
| **Benefit** | **Explanation** |
| Reduces Repetition | Instead of manually entering each value, data is pulled automatically from another table. |
| Improves Accuracy | Minimizes human error by relying on existing, validated data. |
| Saves Time | Very useful when inserting large volumes of related data. |
| Business Logic | Allows inserting only relevant records — e.g., employees in a certain department or students from a specific city. |

**Normalization**

**Normalization** is the process of organizing data in a database to **minimize redundancy** and **improve data integrity.**

**Goals of Normalization**

✔ **Eliminate duplicate data** (redundancy)  
✔ **Reduce anomalies** (insert, update, delete anomalies)  
✔ **Improve data integrity** (consistent and accurate data)  
✔ **Optimize storage** (efficient use of space)

|  |  |
| --- | --- |
| **Normal Form** | **Rule** |
| **1NF** | All attributes must be atomic (no repeating groups). |
| **2NF** | Must be in 1NF + no partial dependency (non-key columns depend on the whole primary key). |
| **3NF** | Must be in 2NF + no transitive dependency (non-key columns depend only on the primary key). |
| **BCNF** | Stricter than 3NF—every determinant must be a candidate key. |
| **4NF** | No multi-valued dependencies. |

**Output of Normalization**

* A **well-structured database** with **multiple related tables**.
* **Foreign keys** establish relationships.
* **Minimal redundancy** and **better query performance** (in OLTP systems).

**Encoding System**

**An encoding system defines how to represent characters as numbers (bytes) so computers can store and process them.**

**Here are the most common encoding systems:**

**🔹 ASCII (American Standard Code for Information Interchange):**

* **Uses 7 bits (128 values).**
* **Supports only English characters: A–Z, a–z, 0–9, punctuation.**
* **Example: 'A' = 65, 'a' = 97**

**🔹 Extended ASCII / ISO 8859-1:**

* **Uses 8 bits (256 values).**
* **Adds some extra European symbols (like é, ñ).**

**🔹 Unicode:**

* **Supports all languages and many symbols (emoji, math symbols, etc.).**
* **Has several encoding forms:**
  + **UTF-8: Variable length (1–4 bytes), most popular worldwide.**
  + **UTF-16: 2 or 4 bytes.**
  + **UTF-32: 4 bytes for every character.**
* **Example:**
  + **'A' in UTF-8 = 0x41**
  + **'أ' (Arabic letter) in UTF-8 = 0xD8 0xA3**

**EX 🡪 MAX(Name)**

**To determine which name is the "maximum," SQL compares characters in each name based on their encoded numeric values.**

**Every character (like A, b, أ) has a corresponding numeric value in an encoding system like ASCII or UTF-8.**

**SQL يقارن "Bob" و "Charlie" و "Alice":**

* **هو مش بيقارن بالحروف كحروف.**
* **هو بيقارن بالأرقام اللي الحروف دي بتمثلها في الـ encoding system**

**Self join if relationship n:m**

1. Creating the Junction Table

CREATE TABLE StudentSupervisor (

StudentID INT,

SupervisorID INT,

PRIMARY KEY (StudentID, SupervisorID),

FOREIGN KEY (StudentID) REFERENCES Students(St\_ld),

FOREIGN KEY (SupervisorID) REFERENCES Students(St\_ld)

)

* StudentID: References the student being supervised
* SupervisorID: References the supervising student
* Composite primary key ensures unique relationships

2. Populating the Junction Table

INSERT INTO StudentSupervisor (StudentID, SupervisorID)

SELECT St\_ld, St\_super

FROM Students

WHERE St\_super IS NOT NULL

3. Querying with Self Join

SELECT s1.St\_ld AS StudID , s1.St\_Fname AS StudName , s2.St\_ld AS SupID, s2.St\_Fname AS SupName

FROM Students s1

JOIN StudentSupervisor ss ON s1.St\_ld = ss.StudentID

JOIN Students s2 ON ss.SupervisorID = s2.St\_ld

--------------------------------------------------------------------------------------------------------------------------------

Delete with subquery + join

delete SC /// without subquery

from Stud\_Course SC,student S

where S.st\_id=SC.st\_id and s.st\_address = 'cairo'

delete from Stud\_Course /// with subquery

where St\_Id IN (

select St\_Id

from Student

where St\_Address = 'cairo'

)