National University of Computer and Emerging Sciences

Laboratory Manuals

for

Database Systems Lab

(CL -2005)

Course Instructor	Ms. Aleena Ahmad
Lab Instructor	Seemab Ayub
Lab TA	Amal Sarmad
Section	BCS-4E
Semester	Spring 2025

Department of Computer Science
FAST-NU, Lahore, Pakistan

Lab Manual 07

SQL

SQL tutorial gives unique learning on Structured Query Language and it helps to make practice on SQL commands which provides immediate results. SQL is a language of database, it includes database creation, deletion, fetching rows and modifying rows etc. SQL is an ANSI (American National Standards Institute) standard, but there are many different versions of the SQL language.

Why SQL?

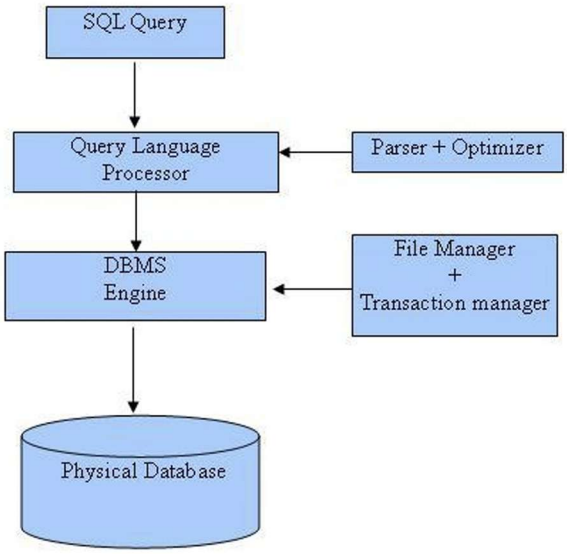
- Allows users to access data in relational database management systems.
- Allows users to describe the data.
- Allows users to define the data in the database and manipulate that data.
- Allows embedding within other languages using SQL modules, libraries & pre-compilers.
- Allows users to create and drop databases and tables.
- Allows users to create views, stored procedure, functions in a database.
- Allows users to set permissions on tables, procedures and views

SQL Process

When you are executing an SQL command for any RDBMS, the system determines the best way to carry out your request and SQL engine figures out how to interpret the task.

There are various components included in the process. These components are Query Dispatcher, Optimization Engines, Classic Query Engine and SQL Query Engine, etc. Classic query engine handles all non-SQL queries, but SQL query engine won't handle logical files.

Following is a simple diagram showing SQL Architecture:



SQL Commands

In previous manual we covered DDL & DML, in this manual we'll cover DQL

DQL - Data Query Language:

Command	Description
SELECT	Retrieves certain records from one or more tables

Details of DQL

---

1. Introduction to Triggers

A **trigger** is a special type of stored procedure that automatically executes when a specific event occurs in a database. Triggers enforce business rules, maintain data integrity, and automate tasks such as logging, validation, and auditing.

Triggers are primarily classified into:

- **DML (Data Manipulation Language) Triggers** → Fires on `INSERT`, `UPDATE`, or `DELETE` operations.
- **DDL (Data Definition Language) Triggers** → Fires on schema changes like `CREATE`, `ALTER`, or `DROP`.
- **Logon Triggers** → Fires when a user logs into SQL Server.

---

2. Types of Triggers

2.1. DML Triggers

DML triggers are triggered by `INSERT`, `UPDATE`, or `DELETE` operations. They are further divided into:

1. **AFTER Triggers (For/After Triggers)**
2. **INSTEAD OF Triggers**
3. **Nested Triggers (Triggers Calling Other Triggers)**
4. **Recursive Triggers (Self-Calling Triggers)**

2.2. DDL Triggers

Fires when schema-related events occur (`CREATE`, `ALTER`, `DROP`).

2.3. Logon Triggers

Fires when a user session is established in SQL Server.

---

3. DML Trigger Types with Examples (Teacher-Student Scenario)

3.1. AFTER Triggers (FOR/AFTER TRIGGERS)

Definition:

- Executes after the triggering `INSERT`, `UPDATE`, or `DELETE` statement is successfully executed.
- Can be used to enforce business rules, logging, or maintaining historical data.

Example: AFTER INSERT Trigger (Logging Student Registrations)

```sql
-- Create an AFTER INSERT trigger to log student enrollment
CREATE TRIGGER trg_AfterInsert_Student
ON Students
AFTER INSERT
AS
BEGIN
    INSERT INTO Student_Log (student_id, log_message)
    SELECT student_id, 'New student enrolled: ' + name
    FROM inserted;
END;
```

3.2. INSTEAD OF Triggers

Definition:

- Executes instead of the `INSERT`, `UPDATE`, or `DELETE` operation.
- Used when direct changes to a table should be prevented or controlled.

Example: Preventing Direct Deletion of Students

```sql
-- Create an INSTEAD OF DELETE trigger to prevent deletion of students
CREATE TRIGGER trg_PreventDelete_Student
ON Students
INSTEAD OF DELETE
AS
BEGIN
    PRINT 'Error: You cannot delete a student record directly!';
END;
```

3.3. Nested Triggers

Definition:

- When a trigger performs an action that causes another trigger to execute.

- SQL Server allows triggers to be nested up to **32 levels**.

Example: Auto-Update Course Enrollment When Student Registers

```
-- Trigger to enroll a student in a default course upon registration
CREATE TRIGGER trg_AutoEnroll_Student
ON Students
AFTER INSERT
AS
BEGIN
    INSERT INTO Student_Courses (student_id, course_id)
    SELECT student_id, 1  -- Assume course_id = 1 is a default course
    FROM inserted;
END;
```

```
-- Trigger to increment the course's enrolled_students count
CREATE TRIGGER trg_UpdateCourseEnrollment
ON Student_Courses
AFTER INSERT
AS
BEGIN
    UPDATE Courses
    SET enrolled_students = enrolled_students + 1
    WHERE course_id IN (SELECT course_id FROM inserted);
END;
```

3.4. Recursive Triggers

Definition:

- A trigger calls itself either **directly** or **indirectly** through another trigger.

Example: Preventing Unauthorized Student Age Updates

```
-- Enable recursive triggers (optional)
EXEC sp_configure 'nested triggers', 1;
RECONFIGURE;
```

```sql
-- Create a trigger that prevents students' age from being reduced
CREATE TRIGGER trg_PreventAgeReduction
ON Students
AFTER UPDATE
AS
BEGIN
    IF EXISTS (
        SELECT 1 FROM inserted i
        JOIN deleted d ON i.student_id = d.student_id
        WHERE i.age < d.age
    )
    BEGIN
        PRINT 'Error: Age cannot be decreased!';
        ROLLBACK TRANSACTION;
    END;
END;
```

---

4. DDL Triggers (Schema-Level Triggers)

Definition:

- Fires on schema changes (`CREATE`, `ALTER`, `DROP`).
- Used for auditing schema modifications.

Example: Preventing Table Deletion

```sql
CREATE TRIGGER trg_PreventTableDrop
ON DATABASE
FOR DROP_TABLE
AS
BEGIN
    PRINT 'Error: Table deletion is not allowed!';
    ROLLBACK TRANSACTION;
END;
```

5. Logon Triggers

Definition:

- Fires when a user logs into the database.
- Used for security enforcement.

Example: Restricting Login After Business Hours

```
CREATE TRIGGER trg_RestrictLogon
ON ALL SERVER
FOR LOGON
AS
BEGIN
    IF DATEPART(HOUR, GETDATE()) NOT BETWEEN 9 AND 18
    BEGIN
        PRINT 'Error: Logins are only allowed between 9 AM and 6 PM.';
        ROLLBACK;
    END;
END;
```

`ROLLBACK TRANSACTION` in SQL Server

1. Definition:

`ROLLBACK TRANSACTION` is a SQL command used to undo changes made in the current transaction, restoring the database to its previous state before the transaction began.

2. Use Cases:

- Error handling → Prevents invalid or incomplete data modifications.
- Trigger-based validation → Reverts actions when business rules are violated.
- Transactional consistency → Ensures atomicity (all or nothing) in operations.

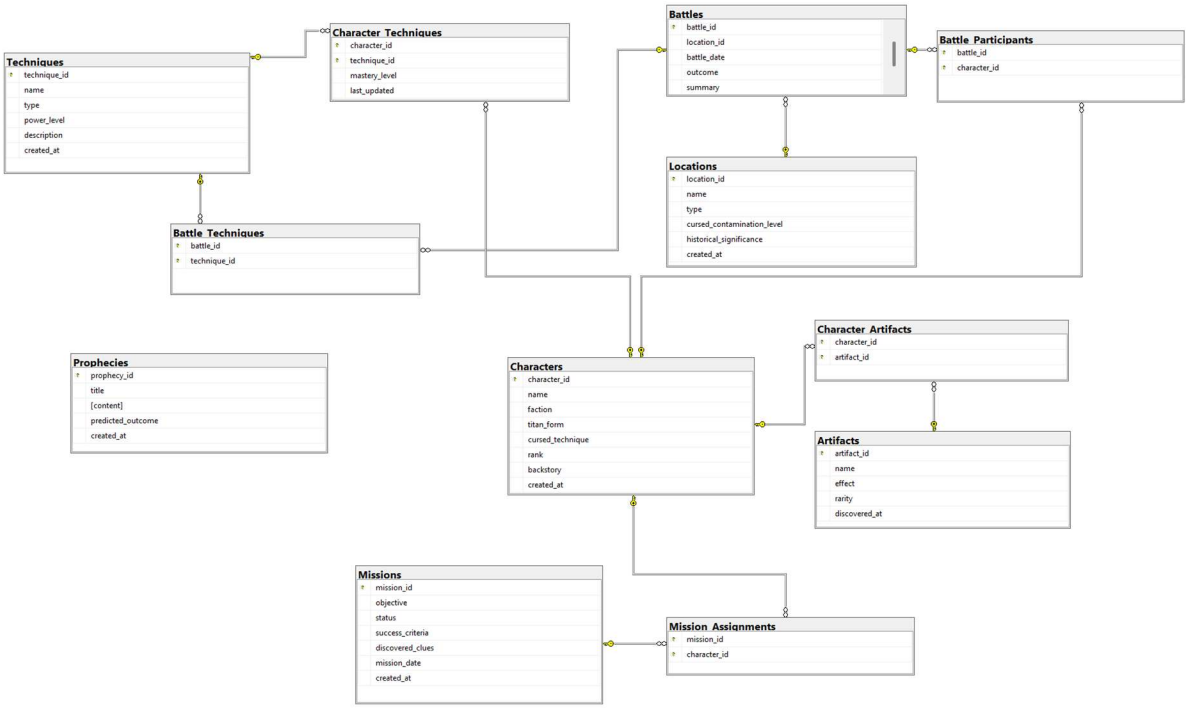
Advantages of Triggers

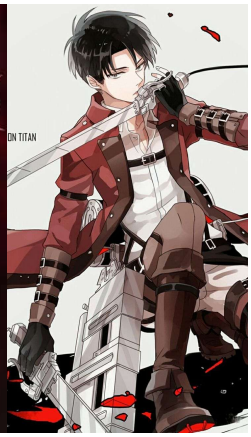
- **Ensures Data Integrity** – Automatically enforces business rules and constraints.
- **Automates Complex Tasks** – Executes actions like logging, updates, and calculations.

- **Enforces Business Rules** – Ensures consistency across all data modifications.
- **Maintains Audit Trails** – Logs changes for tracking and accountability.
- **Prevents Unauthorized Modifications** – Restricts critical actions like deletions or schema changes.
- **Automates Referential Actions** – Handles cascading updates and deletions.
- **Enhances Security** – Restricts access based on conditions like time or user roles.
- **Reduces Application Complexity** – Moves logic to the database, reducing code duplication.
- **Handles Derived Data** – Updates summary tables and computed fields automatically.
- **Supports Event-Driven Workflows** – Triggers can notify or react to database changes.

Happy learning and querying!

Download sql file provided with this manual. Run that file and a schema with following details will be created:





In Lab Exercises

The Chronicles of Mystical Adventures

In a dystopian future where humanity teeters on the brink of extinction, a mysterious phenomenon has merged two horrifying realities: the relentless Titans and malevolent cursed spirits. With the collapse of the old world order, the remnants of human civilization have built massive defensive Walls. Beyond these barriers, monstrous Titans now roam—beings that, in an uncanny twist of fate, are also influenced by cursed energy. To battle this dual threat, the Survey Corps has joined forces with a secretive order of jujutsu sorcerers. Their mission: to unlock ancient techniques that merge Titan transformation with cursed energy manipulation in order to defeat their foes and uncover the truth behind this sinister convergence.

Scenario Background:

- **Dual Origins:**
  Some humans have inherited the power to transform into Titans, a gift once thought to be a curse. Simultaneously, a rare few possess the ability to harness jujutsu—a technique passed down to fight curses. Now, there are warriors who straddle both worlds, wielding Titan might alongside potent cursed energy.

- **Ancient Secrets:**
  Rumors persist of a forbidden ritual that can permanently sever the link between Titans and curses, but to execute it, the alliance must decode cryptic prophecies, recover lost artifacts, and understand the intertwined lineage of cursed sorcery and Titan heritage.

- **Battle Records:**
  Fierce battles rage at cursed sites around the Walls, where titanic curses and jujutsu techniques collide. The outcomes of these battles, along with detailed records of techniques, character evolutions, and strategic alliances, are essential to unraveling the mystery and predicting future threats.

Schema:

A relational database schema that captures the following aspects of this cross-over universe:

- **Character Profiles:**
  Track individuals (both Titan-shifters and jujutsu sorcerers) with their unique abilities, transformation records, cursed technique mastery levels, and personal histories. Some characters may even belong to both categories, and your schema must reflect this overlap.

- **Battles and Alliances:**
  Record detailed battle events including location, date, participating characters, employed techniques (both Titan roars and jujutsu incantations), and outcomes. Model many-to-many relationships between characters and battles, ensuring that you can query which alliances formed during key encounters.

- **Locations and Events:**
  Maintain information about critical sites such as the Walls, cursed shrines, and battlefield coordinates. Each location could be linked to recurring events (e.g., breaches, curse manifestations) and have attributes like historical significance or level of cursed contamination.

- **Artifacts and Techniques:**
  Log ancient artifacts that enhance or neutralize cursed energy, as well as a catalog of jujutsu techniques and Titan abilities. Ensure that your database can track which characters have mastered specific techniques, and how these abilities evolve over time.

- **Prophecies and Missions:**
  Include a component for mysterious prophecies that hint at the origins of the Titan curse. Additionally, model missions assigned to teams that combine jujutsu sorcerers and Titan-shifters—storing objectives, assigned personnel, success criteria, and any discovered clues about the forbidden ritual.

Kindly help them writing appropriate Triggers to save the world

1. Log new **Titan-shifter or Jujutsu Sorcerer** additions in a Chracters_Log table. (create table and insert from trigger)

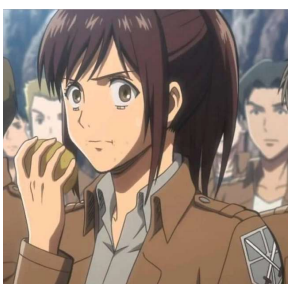
2. When a character's **rank** changes, log the update. (Hint: create Rank_log table)

3. Prevent deletion of **Gojo Satoru** from the Characters table.

4. Prevent inserting **Jujutsu Sorcerers** with a **power level below 30**.

5. Prevent **a warrior's technique mastery level from being reduced**.

6. Prevent accidental deletion of important tables like **Characters**.



7. *Track changes to all tables (`CREATE`, `ALTER`, `DROP`). (Hint: Trigger on DB)*

8. *Only allow logins between **9 AM and 6 PM**. (Don't execute this trigger query, just write code)*

9. *Whenever a battle occurs, track the **strongest Hybrid warrior** (highest `mastery_level`) and store their name in the **Battle_Strongest_Hybrid** table.*

Extra Exercises & Project Practice

1. Audrey and Klein are looking for **missions where "At Risk" or "Lost Control" Beyonders are involved AND a high-risk artifact is connected.**

2. Connect to SQL Server using connection string from NodeJS and insert/retrieve missions table

3. Write NodeJS endpoints to retrieve data from database. For example, /missions to get all missions

---

Submission Guidelines

---

1. submit following files strictly following the naming convention: l231234.sql

---

Best of Luck! Happy Querying
