National University of Computer and Emerging Sciences

Laboratory Manuals

for

Database Systems Lab

(CL -2005)

Course Instructor	Ms. Aleena Ahmad
Lab Instructor	SeemabAyub
Section	BCS-4E
Semester	Spring 2025

Department of Computer Science
FAST-NU, Lahore, Pakistan

Lab Manual 05

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?

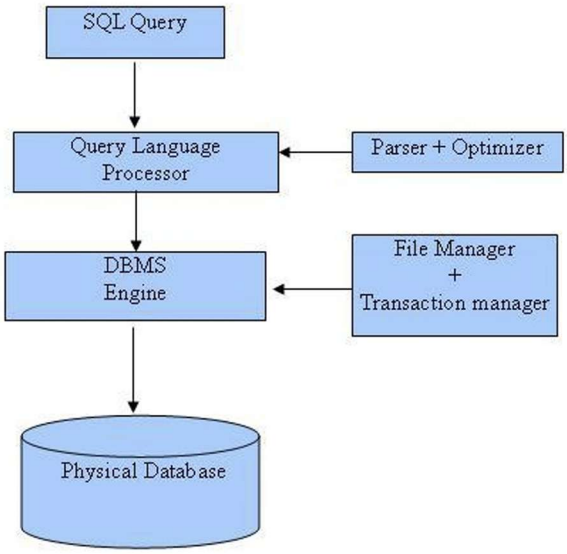
- Allows users to access data in relational database management systems.
- Allows users to describe the data.
- Allows users to define the data in the database and manipulate that data.
- Allows embedding within other languages using SQL modules, libraries & pre-compilers.
- Allows users to create and 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. Views in SQL Server

Definition

A **view** in SQL Server is a **virtual table** that contains the result of a SQL query. It does not store data physically but dynamically retrieves data from underlying tables whenever accessed.

Advantages of Views

- **Data Abstraction**: Simplifies query complexity by presenting a virtual table.
- **Security**: Restricts access to specific columns or rows.
- **Reusability**: Saves frequently used queries for easy access.
- **Performance Optimization**: Improves performance by pre-compiling complex queries.

Syntax for Creating a View

```
CREATE VIEW view_name AS

SELECT column1, column2, ...

FROM table_name

WHERE condition;
```

Example

Creating a view that shows employees' names and salaries from the Employees table:

```
CREATE VIEW EmployeeSalaries AS

SELECT EmployeeID, FirstName, LastName, Salary

FROM Employees

WHERE Salary > 50000;
```

Updating Data Through a View

A view can be **updated** if it meets certain conditions:

1. The view is based on **a single table**.
2. It does not use **aggregations** (SUM, AVG, etc.).
3. It does not use **DISTINCT** or **GROUP BY**.
4. It does not contain **computed columns** or **expressions**.
5. It does not involve **JOINs on multiple tables** (except updateable views with INSTEAD OF triggers).

Example of updating a view:

```
UPDATE EmployeeSalaries

SET Salary = 70000

WHERE EmployeeID = 101;
```

This will update the underlying Employees table if conditions are met.

When a View Cannot Be Updated

A view **cannot be updated** if it:

❌ Contains **aggregations** (e.g., SUM, AVG, MAX, MIN).
❌ Uses **GROUP BY** or **DISTINCT**.
❌ Contains **JOINs** on multiple tables without an INSTEAD OF trigger.
❌ Uses **computed columns** or **expressions** (e.g., Salary * 1.1).
❌ References a **non-updatable view**.

Example of a non-updatable view:

```
CREATE VIEW TotalSalaries AS

SELECT DepartmentID, SUM(Salary) AS TotalSalary

FROM Employees

GROUP BY DepartmentID;
```

Trying to update TotalSalaries will **fail** because it contains an aggregation (SUM).

Modifying a View

```
ALTER VIEW view_name AS

SELECT column1, column2, ...

FROM table_name

WHERE condition;
```

Example:

```
ALTER VIEW EmployeeSalaries AS

SELECT EmployeeID, FirstName, LastName, Salary, DepartmentID

FROM Employees

WHERE Salary > 60000;
```

Dropping a View

```
DROP VIEW view_name;
```

Example:

```
DROP VIEW EmployeeSalaries;
```

---

2. Stored Procedures in SQL Server

Definition

A **stored procedure** is a precompiled collection of **SQL statements and logic** stored in the database. It can accept parameters and return values.

Advantages of Stored Procedures

- **Performance Improvement**: Precompiled and cached for faster execution.
- **Security**: Restricts direct access to tables.
- **Code Reusability**: Can be called multiple times from different applications.
- **Reduced Network Traffic**: Executes on the database server, reducing the amount of data sent over the network.

Syntax for Creating a Stored Procedure

```
CREATE PROCEDURE procedure_name

AS

BEGIN

   -- SQL statements

END;
```

Example

Creating a stored procedure to fetch employees with a salary above a given amount:

```
CREATE PROCEDURE GetHighSalaryEmployees

   @MinSalary DECIMAL(10,2)

AS

BEGIN

   SELECT EmployeeID, FirstName, LastName, Salary

   FROM Employees

   WHERE Salary > @MinSalary;

END;
```

Executing a Stored Procedure

```
EXEC procedure_name;
```

Example:

```
EXEC GetHighSalaryEmployees @MinSalary = 60000;
```

Stored Procedure with Output Parameter

```
CREATE PROCEDURE GetTotalEmployees
    @TotalCount INT OUTPUT
AS
BEGIN
    SELECT @TotalCount = COUNT(*) FROM Employees;
END;
```

Executing and retrieving the output:

```
DECLARE @EmpCount INT;
EXEC GetTotalEmployees @EmpCount OUTPUT;
PRINT @EmpCount;
```

Modifying a Stored Procedure

```
ALTER PROCEDURE procedure_name
AS
BEGIN
    -- Updated SQL statements
END;
```

Example:

```
ALTER PROCEDURE GetHighSalaryEmployees
    @MinSalary DECIMAL(10,2),
    @DepartmentID INT
AS
BEGIN
    SELECT EmployeeID, FirstName, LastName, Salary
    FROM Employees
    WHERE Salary > @MinSalary AND DepartmentID = @DepartmentID;
END;
```

Dropping a Stored Procedure

```
DROP PROCEDURE procedure_name;
```

Example:

```
DROP PROCEDURE GetHighSalaryEmployees;
```

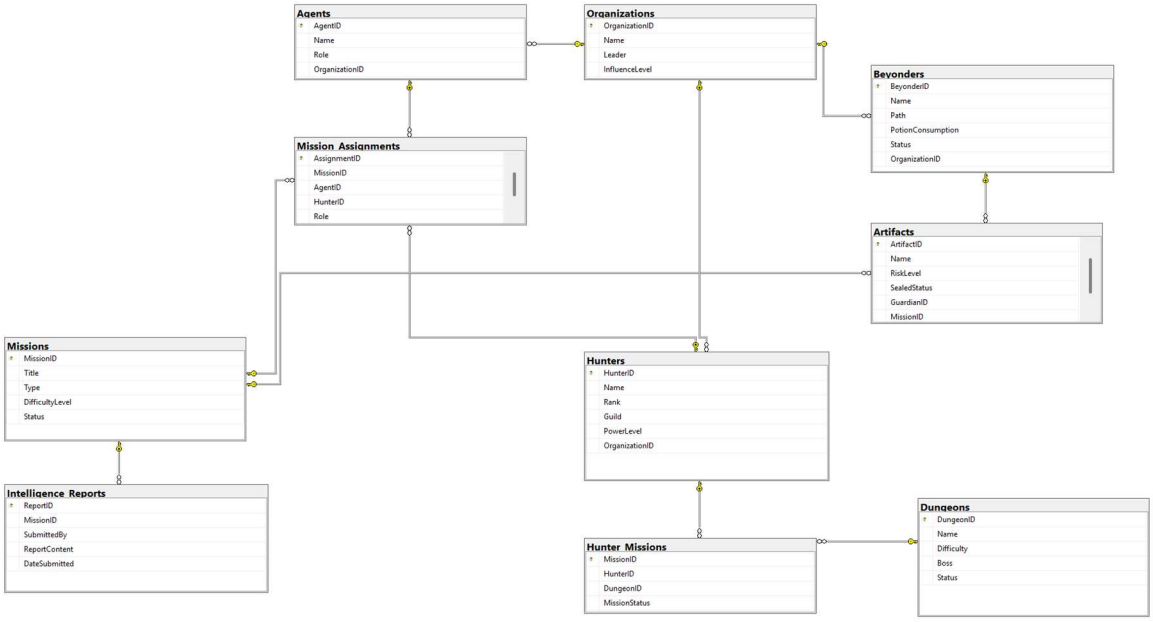
Comparison: Views vs. Stored Procedures

Feature	Views	Stored Procedures
Purpose	Provides a virtual table for querying data	Contains SQL logic and can perform complex operations
Data Storage	Does not store data; retrieves it dynamically	Stores precompiled SQL logic but not actual data

Can Accept Parameters	❌ No	✅ Yes
Can Modify Data	❌ No (unless an updateable view)	✅ Yes
Performance	Faster for read-only operations	Faster for complex queries and business logic execution
Security	Hides underlying table structure	Restricts direct table access

Happy learning and querying!

Download lab-5 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 world where dungeons breach reality, artifacts hold unspeakable power, and Beyonders manipulate fate, a silent war rages beneath the surface.

Sung Jin-Woo, the Shadow Monarch, leads the Celestial Hunters, taming the abyss one dungeon at a time. His strength is unmatched, yet whispers of an ancient force—one that predates even the Monarchs—begin to haunt his shadows.

Meanwhile, Audrey Hall, a rising star of the Church of Evernight, senses disturbances beyond comprehension. Visions of hidden cults, lost artifacts, and unseen hands guiding destiny consume her mind. The Nighthawks deploy their finest agents, but every lead only deepens the mystery.

In the heart of Backlund, Klein Moretti, the Fool who watches from above, weaves his own web. He knows that the dungeons, Beyonders, and secret organizations are mere pieces of a far greater puzzle—one that threatens to rewrite reality itself.

The Hunters raid dungeons, the Agents uncover conspiracies, and the Beyonders chase power, but in the end, only one truth remains:

🔥 He who controls the database… controls the fate of the world.

Kindly help them writing appropriate views and stored procedures to save the world

1. Sung Jin-Woo needs a list of **all dungeons he has entered**, including their difficulty and current status. **Create a View that shows** `DungeonID`, `Name`, `Difficulty`, `Boss`, `Status` and Only includes dungeons where **Jin-Woo (HunterID = 1) has participated**.

2. Audrey Hall suspects **several Beyonders** are losing control and might pose a threat. She needs a report on **all Beyonders who are "At Risk" or "Lost Control"**, along with their organization. **Create a View that shows** `BeyonderID`, `Name`, `Path`, `PotionConsumption`, `Status`, `OrganizationName`

3. Klein Moretti is monitoring all **ongoing and pending** missions across different organizations. **Create a View that shows** `MissionID`, `Title`, `Type`, `DifficultyLevel`, `Status`, `OrganizationName`

4. **Audrey Hall** is searching for **missions that have repeatedly failed**, especially when high-risk Beyonders were involved. She needs a **summary of the most dangerous missions where multiple agents/hunters have failed**, sorted by failure count. **Create a View that displays** `MissionID`, `Title`, `TimesFailed`, `MostCommonHunter`, `MostCommonBeyonder`, `HighRiskArtifact` and If a **High-Risk Artifact** was linked, display its **name**.

5. Sherlock Holmes wants a **quick list of the strongest S-Rank Hunters** for an elite mission. **Create a View that shows** `HunterID`, `Name`, `Guild`, `PowerLevel`, `OrganizationName` and Only include hunters with **Rank = 'S'**

6. Go Gunhee wants a **procedure** to assign a **Hunter to a Dungeon** but only if they are **not already assigned**. Create a procedure which takes two inputs (hunterid, dungeonid) and assigns the specified hunter to the dungeon if not already assigned.

7. *Sung Jin-Woo requests a **procedure** to **mark dungeons as "Cleared"** once all assigned hunters have completed their mission.*

8. *Audrey Hall wants **automatic logging** whenever an artifact is retrieved/recovered. Create a procedure to insert info in intelligence reports. For example, INSERT INTO Intelligence_Reports*

(MissionID, SubmittedBy, ReportContent, DateSubmitted) VALUES (@MissionID, 'Audrey Hall', 'Artifact successfully retrieved and sealed.', GETDATE());

9. *Go Gunhee requires **hunters to be promoted automatically** when their **PowerLevel increases above thresholds**. Create a procedure which takes a hunter id and sets rank as per following*
    a. *Rank 'S' when power >= 9000*
    b. *Rank 'A' when power >= 7500*
    c. *Rank 'B' when power >= 5000*
10. *Klein Moretti has intercepted multiple failed missions related to the Necronomicon and other dark artifacts. He needs a procedure that:*
    a. *Identifies agents/hunters who repeatedly failed artifact-related missions.*
    b. *Logs them as "Forbidden Knowledge" suspects in the Intelligence Reports.*
    c. *Stored Procedure Requirements:*
        i. *If an agent/hunter failed at least 3 missions involving high-risk artifacts, log their details.*
        ii. *Insert a warning into Intelligence Reports.*

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