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 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?

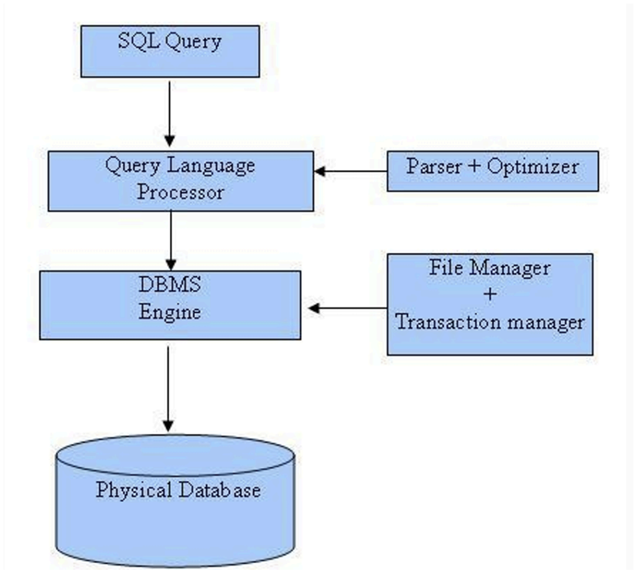
- Allows users to access data in relational database management systems.
- Allows users to describe the data.
- Allows users to define the data in the database and manipulate that data.
- Allows embedding within other languages using SQL modules, libraries & pre-compilers.
- Allows users to create and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

SQL subqueries enable complex queries by performing multiple steps, such as filtering data based on aggregated values or computing derived results.

1.1 Subquery in WHERE Clause

A subquery in the `WHERE` clause allows filtering results based on another query.

Example Scenario:
Find students who have paid the highest tuition fee.

```
SELECT student_name, tuition_fee
FROM Payments
WHERE tuition_fee = (SELECT MAX(tuition_fee) FROM Payments);
```

Explanation:

- The inner query `(SELECT MAX(tuition_fee) FROM Payments)` fetches the highest tuition fee.
- The outer query retrieves students who have paid this fee.

1.2 Subquery in FROM Clause

A subquery in the `FROM` clause allows treating its result as a temporary table.

Example Scenario:
Find the average tuition fee of students who have paid more than $10,000.

```
SELECT AVG(high_payers.tuition_fee) AS avg_high_payers
FROM (SELECT tuition_fee FROM Payments WHERE tuition_fee > 10000) AS high_payers;
```

Explanation:

- The subquery `(SELECT tuition_fee FROM Payments WHERE tuition_fee > 10000)` filters students who paid more than $10,000.
- The outer query calculates their average tuition fee.

1.3 Subquery in SELECT Clause

A subquery in the `SELECT` clause allows computing derived values.

Example Scenario:
List all students along with the highest tuition fee paid.

```
SELECT student_name, tuition_fee,
    (SELECT MAX(tuition_fee) FROM Payments) AS highest_fee
FROM Payments;
```

Explanation:

- The subquery `(SELECT MAX(tuition_fee) FROM Payments)` calculates the highest tuition fee.
- The outer query lists all students along with this value.

---

2. Correlated Subqueries

A correlated subquery references columns from the outer query and executes once per row.

2.1 Finding Students Who Paid Above the Average Fee

```
SELECT student_name, course_name, tuition_fee  FROM Enrollments e1

WHERE tuition_fee = (SELECT MAX(tuition_fee) FROM Enrollments e2 WHERE e1.student_id =
e2.student_id);
```

Explanation:

- The subquery finds the highest tuition fee per student.
- The outer query retrieves the corresponding course.

---

3. Advanced MSSQL Functions with Subqueries

3.1 Using ROW_NUMBER() to Rank Students by Tuition Fee

```
SELECT student_name, tuition_fee,

    ROW_NUMBER() OVER (ORDER BY tuition_fee DESC) AS rank

FROM Payments;
```

Explanation:

- `ROW_NUMBER()` assigns a unique rank to each student based on the tuition fee.
- Students with the highest fees are ranked first.

3.2 Using PARTITION BY to Calculate Running Totals

```
SELECT student_name, tuition_fee,

    SUM(tuition_fee) OVER (PARTITION BY course_id ORDER BY student_name) AS running_total

FROM Payments;
```

Explanation:

- `SUM() OVER (PARTITION BY course_id ORDER BY student_name)` calculates a running total of tuition fees per course.
- Each row shows the cumulative sum of fees paid by students in a course.

3.3 Using CTEs with Recursive Queries

Example Scenario:
Find a hierarchical list of course prerequisites.

```
WITH CourseHierarchy AS (

  SELECT course_id, prerequisite_id

  FROM CoursePrerequisites

  WHERE prerequisite_id IS NULL
```

```
    UNION ALL

    SELECT c.course_id, ch.prerequisite_id

    FROM CoursePrerequisites c

    INNER JOIN CourseHierarchy ch ON c.prerequisite_id = ch.course_id

)

SELECT * FROM CourseHierarchy;
```

Explanation:

- A Common Table Expression (CTE) recursively retrieves course prerequisites.
- The `UNION ALL` statement enables traversing hierarchical relationships.

3.4 Using LEAD() and LAG() for Comparative Analysis

```
SELECT student_name, tuition_fee,
       LAG(tuition_fee) OVER (ORDER BY tuition_fee DESC) AS previous_fee,
       LEAD(tuition_fee) OVER (ORDER BY tuition_fee DESC) AS next_fee
FROM Payments;
```

Explanation:

- `LAG()` fetches the previous row's tuition fee.
- `LEAD()` fetches the next row's tuition fee.
- This allows comparisons between consecutive rows.

3.5 Using DENSE_RANK() for Ranking Without Gaps

```
SELECT student_name, tuition_fee,
       DENSE_RANK() OVER (ORDER BY tuition_fee DESC) AS rank
FROM Payments;
```

Explanation:

- **`DENSE_RANK()` ranks students by tuition fee, ensuring consecutive ranking numbers without gaps.**

3.6 Using NTILE() for Percentile Distribution

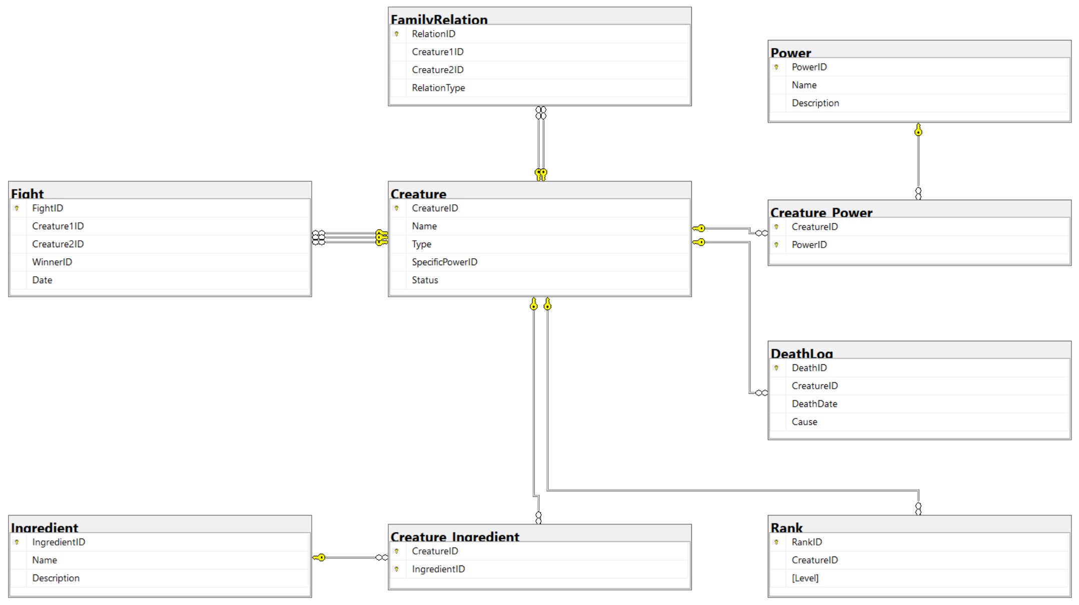
```
SELECT student_name, tuition_fee,

    NTILE(4) OVER (ORDER BY tuition_fee DESC) AS quartile
FROM Payments;
```

Explanation:

- `NTILE(4)` divides students into four quartiles based on tuition fee distribution.

```
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

PowerPuffDB: The Battle Records

In Townsville, battles between heroes and villains shape history. PowerPuffDB is the sacred archive tracking every hero, villain, power, and fight.

The Rules of Power

- Bloodlines Cannot Be Broken 🩸 – Family ties must be preserved, preventing automatic deletions.
- Fights Must Be Logged ⚔️ – Every battle is recorded, but victors cannot vanish without a trace.
- Balance is Key ⚖️ – Powers and ranks follow strict rules, ensuring fairness in the chaos.

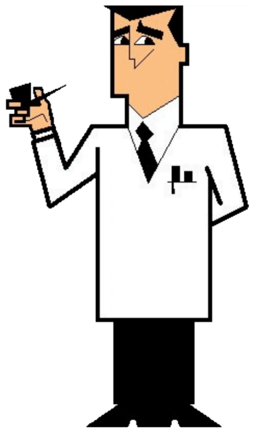
As the Guardian of PowerPuffDB, your mission is to uphold order. Code wisely—history depends on it. Help writing the following queries using **ONLY NESTED/SUBQUERIES**.

1. *Find the name of the strongest creature (highest rank).*
2. *Get the most frequently used ingredient.*
3. *Retrieve the creature that fought the most battles.*
4. *Find creatures with only one power.*
5. *Retrieve creatures that have never lost a fight.*
6. *Get the youngest sibling (highest CreatureID).*
7. *Find the creature with the most powers.*
8. *Get creatures that fought at least twice.*
9. *Find creatures that died and had a rank.*

10. *Find creatures that used 'Chemical X' in their creation.*

11. *Find the creature that has fought against the most different opponents.*

Extra Exercises & Project Practice



1. Find the villain with the most family members who were defeated in battle.
2. Find the strongest dead hero by counting their wins before death.

2. Connect to SQL Server using connection string from NodeJS and insert a few records to a table using POST request

3. Write NodeJS endpoints to retrieve data from database. For example, /creatures to get all creatures with their powers

Submission Guidelines

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

Best of Luck! Happy Querying