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	Abdullah Naeem
Section	BCS-4E
Semester	Spring 2025

Department of Computer Science
FAST-NU, Lahore, Pakistan

Lab Manual 04

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?

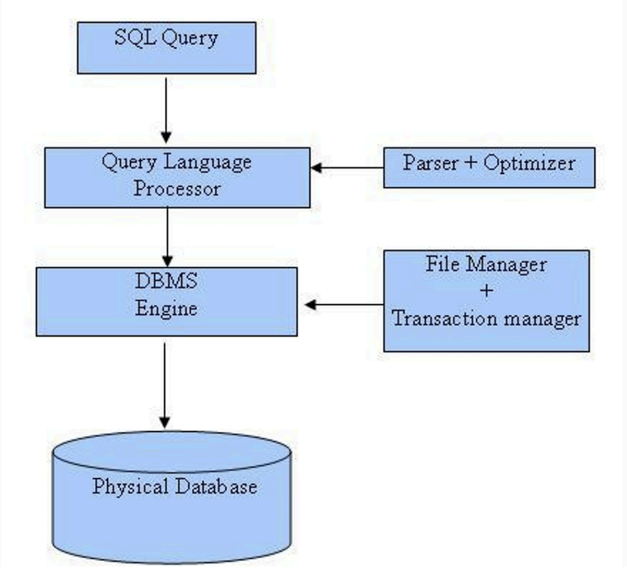
- Allows users to access data in relational database management systems.
- Allows users to describe the data.
- Allows users to define the data in the database and manipulate that data.
- Allows embedding within other languages using SQL modules, libraries & pre-compilers.
- Allows users to create and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

Joins allow you to combine data from multiple tables based on related columns. There are different types of joins, and each serves a different purpose.

1.1 INNER JOIN – Matching Records in Both Tables

An **INNER JOIN** retrieves only the records that have matching values in both tables.

Example Scenario:
You have a **Students** table and a **Courses** table. You want to find the names of students and the courses they are enrolled in.

```
SELECT s.student_name, c.course_name
FROM Students s
INNER JOIN Enrollments e ON s.student_id = e.student_id
INNER JOIN Courses c ON e.course_id = c.course_id;
```

Explanation:

- The INNER JOIN ensures that only students who are enrolled in a course appear in the result.
- The Enrollments table links Students and Courses tables.

1.2 LEFT JOIN – Keeping All Records from the First Table

A **LEFT JOIN** returns all records from the first table (left table) and the matching records from the second table (right table). If there is no match, NULL is returned for columns of the second table.

Example Scenario:
You want to find **all students** and the courses they are enrolled in, but you also want to include students who haven't enrolled in any course.

```
SELECT s.student_name, c.course_name
FROM Students s
LEFT JOIN Enrollments e ON s.student_id = e.student_id
LEFT JOIN Courses c ON e.course_id = c.course_id;
```

Explanation:

- Students who are not enrolled in any course will still appear, but their course_name will be NULL.

1.3 RIGHT JOIN – Keeping All Records from the Second Table

A **RIGHT JOIN** is similar to a LEFT JOIN but keeps all records from the second table (right table) instead of the first.

Example Scenario:
You want to find **all courses** and the students who are enrolled in them, but you also want to include courses that have no students enrolled.

```
SELECT c.course_name, s.student_name
FROM Courses c
RIGHT JOIN Enrollments e ON c.course_id = e.course_id
RIGHT JOIN Students s ON e.student_id = s.student_id;
```

Explanation:

- Courses that have no students enrolled will still be included, but their student_name will be NULL.

1.4 FULL OUTER JOIN – Keeping All Records from Both Tables

A **FULL OUTER JOIN** combines the effects of both LEFT and RIGHT joins, including all records from both tables. If there is no match, NULL appears for the missing side.

Example Scenario:
You want to find **all students and all courses**, whether or not they are linked.

```
SELECT s.student_name, c.course_name
FROM Students s
FULL OUTER JOIN Enrollments e ON s.student_id = e.student_id
FULL OUTER JOIN Courses c ON e.course_id = c.course_id;
```

Explanation:

- This query includes students who are not enrolled and courses that have no students.

---

2. Understanding Aggregations

Aggregation functions allow you to perform calculations on a group of values rather than on individual rows.

2.1 COUNT – Counting the Number of Records

Example Scenario:
You want to count the total number of students in the database.

```
SELECT COUNT(*) AS total_students FROM Students;
```

Explanation:

- COUNT(*) counts all rows in the Students table.

Another Scenario:
Find the number of students enrolled in each course.

```
SELECT c.course_name, COUNT(e.student_id) AS num_students
FROM Courses c
LEFT JOIN Enrollments e ON c.course_id = e.course_id
GROUP BY c.course_name;
```

Explanation:

- COUNT(e.student_id) counts the number of students in each course.
- GROUP BY c.course_name groups the results by course.

2.2 SUM – Adding Up Values

Example Scenario:
You want to find the total tuition fees collected from all students.

```
SELECT SUM(tuition_fee) AS total_fees FROM Payments;
```

Explanation:

- SUM(tuition_fee) adds up all tuition fees in the Payments table.

2.3 AVG – Calculating the Average Value

Example Scenario:
You want to find the average tuition fee paid by students.

```
SELECT AVG(tuition_fee) AS avg_fee FROM Payments;
```

Explanation:

- AVG(tuition_fee) calculates the average fee from the Payments table.

2.4 MIN & MAX – Finding Minimum and Maximum Values

Example Scenario:
Find the highest and lowest tuition fees paid by students.

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

Explanation:

- MIN(tuition_fee) finds the lowest fee.
- MAX(tuition_fee) finds the highest fee.

---

3. Understanding Nested Queries (Subqueries)

A **nested query** (subquery) is a query inside another query.

3.1 Subquery in WHERE Clause

Example Scenario:
Find the 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 subquery (SELECT MAX(tuition_fee) FROM Payments) finds the highest tuition fee.
- The main query retrieves students who have paid that amount.

3.2 Subquery in FROM Clause

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 filters students who paid more than $10,000.
- The main query calculates their average tuition fee.

---

4. Using Ranking Functions

Ranking functions assign a rank to each row in a result set based on a specified column.

4.1 RANK – Assigning Rank to Ordered Data

Example Scenario:
Rank students based on tuition fees paid.

```
SELECT student_name, tuition_fee,
    RANK() OVER (ORDER BY tuition_fee DESC) AS rank
```

```
FROM Payments;
```

Explanation:

- RANK() OVER (ORDER BY tuition_fee DESC) assigns a rank where the highest tuition fee gets rank 1.

---

5. Combining Concepts in Complex Queries

5.1 Finding Students Who Paid Above the Average Fee

```
SELECT student_name, tuition_fee
FROM Payments
WHERE tuition_fee > (SELECT AVG(tuition_fee) FROM Payments);
```

Explanation:

- This query finds students who paid above the average fee.

5.2 Find Courses With the Most Enrolled Students

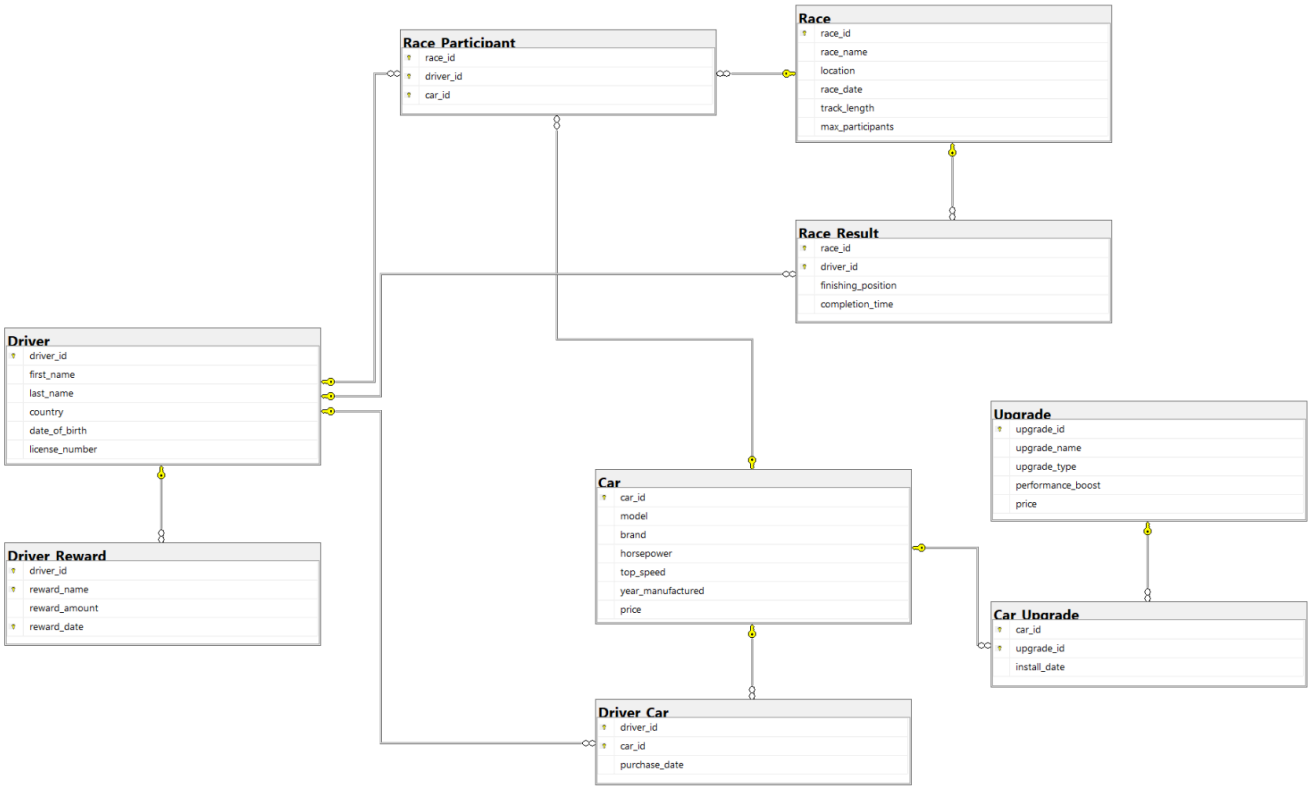
```
SELECT c.course_name, COUNT(e.student_id) AS num_students
FROM Courses c
LEFT JOIN Enrollments e ON c.course_id = e.course_id
GROUP BY c.course_name
ORDER BY num_students DESC
LIMIT 1;
```

Explanation:

- This query counts students per course and returns the course with the most students.

Happy learning and querying!

Download lab-3 sql file provided with this manual. Run that file and a schema with following details will be created:



In Lab Exercises

In the Fast & Furious Racing League, elite racers from around the world compete in high-speed underground races. Each driver owns multiple cars, which can be customized with upgrades. Races happen at different locations, and each race has multiple participants. The winners earn rewards and ranking points. To help DBA to manage functionalities of Fast & Furious, **Write following queries in a SQL file and submit**

Note: First five queries have hint to get you started

1. *List all drivers and their cars.*

   Hint: Use an INNER JOIN between Driver and Driver_Car to link with Car.

2. *Show all race names and the number of participants*

   Hint: Use a LEFT JOIN with COUNT() and GROUP BY.

3. *Display the fastest driver in each race*

   Use a SUBQUERY to find the minimum completion_time per race, then join with Race_Result.

4. *Find the total prize money won by each driver.*

   Hint: Use SUM() and GROUP BY in Driver_Reward.

5. *List all upgrades installed on each car.*

   Use a JOIN between Car, Car_Upgrade, and Upgrade

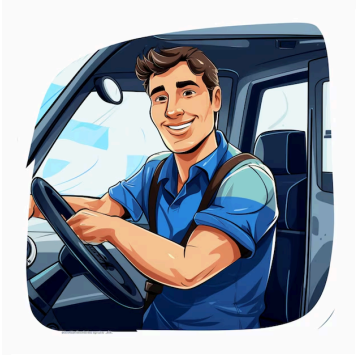
6. *Find drivers who have participated in at least 2 races.*

7. *Display the average completion time for each race.*
8. *Find the driver who owns the most expensive car.*
9. *Show the ranking of drivers based on total prize money won.*
10. *List all drivers who own at least two different car brands.*
11. *Find drivers who have participated in every race.*
12. *Find the top 3 drivers with the fastest average race completion time.*
13. *Display drivers who own cars with at least 2 different upgrades.*

Extra Exercises & Project Practice

---

1. Find the race where the closest finish time difference was recorded
2. Find the driver who won the highest number of races and list their total winnings



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, /sci-fi to get all scifi movies and /sci-fi?after=2010 to get all scifi movies after 2010

Submission Guidelines

---

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

Best of Luck! Happy Querying
