National University of Computer and Emerging Sciences

Laboratory Manuals

for

Database Systems Lab

(CL -2005)

Department of Computer Science
FAST-NU, Lahore, Pakistan

Lab Manual

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?

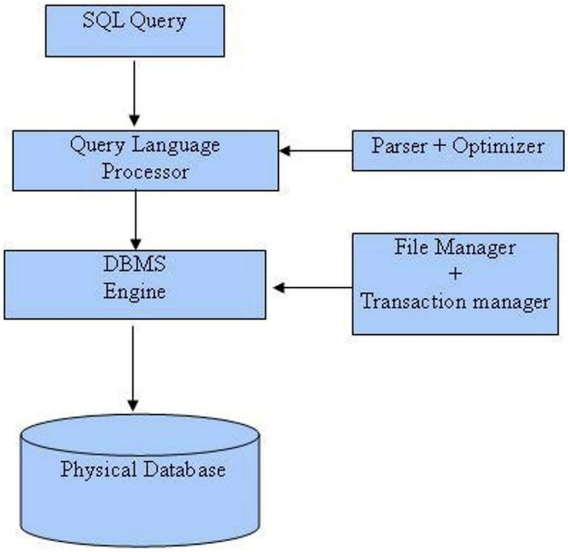
- Allows users to access data in relational database management systems.
- Allows users to describe the data.
- Allows users to define the data in the database and manipulate that data.
- Allows embedding within other languages using SQL modules, libraries & pre-compilers.
- Allows users to create and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



1. CTEs

A CTE (Common Table Expression) is a temporary named result set in SQL that you can reference within a SELECT, INSERT, UPDATE, or DELETE statement.

Think of it as a temporary view that's only available during the execution of a single query.

```
WITH cte_name AS (
    SELECT column1, column2
    FROM some_table
    WHERE condition
)
SELECT *
FROM cte_name;
```

2. PRINT KEYWORD

- Used to display a message in the query output (mainly in SQL Server).
- Helpful for debugging and tracking execution flow inside stored procedures or scripts.

```
PRINT 'I AM A PRINT STATEMENT'
```

3. RAND() FUNCTION

- Returns a random float value between 0 and 1.
- Useful for generating test data or randomizing results.

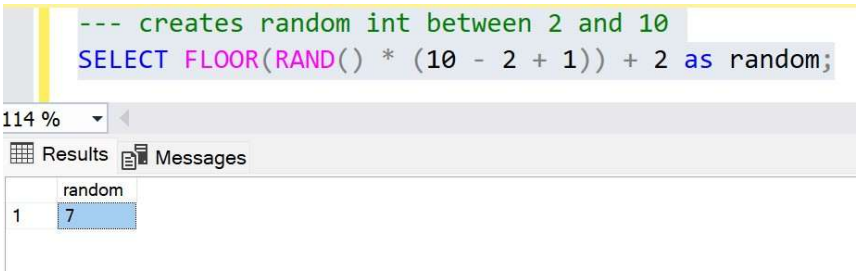
```
select rand() as random
```

167 %

Results | Messages

	random
1	0.657967022393178

- To generate a random integer between two numbers:



PL SQL

PL/SQL stands for Procedural Language extensions to SQL. It is Oracle's procedural programming language, designed to extend the power of SQL with the ability to use programming constructs like variables, loops, conditions, and error handling.

Key Features:

a. Block-structured: Code is written in logical blocks (DECLARE, BEGIN, EXCEPTION, END).

b. Supports control flow: Includes IF, LOOP, WHILE, FOR, CASE, etc.

c. Reusable logic: Write stored procedures, functions, triggers, and packages.

d. Exception handling: Built-in error management using EXCEPTION blocks.

e. Tightly integrated with SQL: Allows use of DML/DDL statements within procedural code.

Handling Variables

- **Declaration**
  - You declare a variable to store and reuse data within SQL blocks, procedures, or functions.

```
DECLARE @x1 INT;
```

- **Setting**

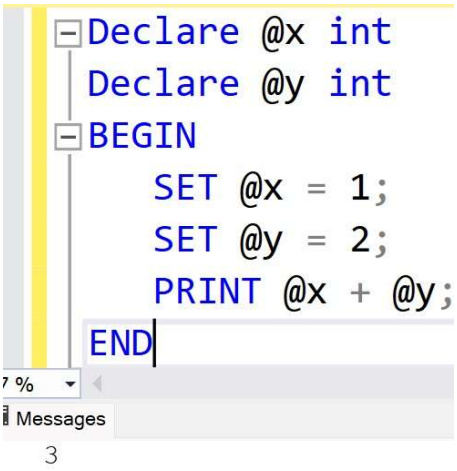
```
SET @count = 5;
```

- **Using**
  - Once declared and set, variables can be used in conditions, calculations, and output.

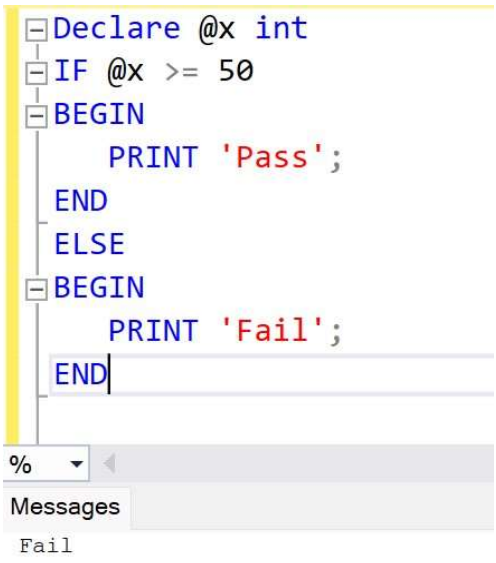
```
DECLARE @count INT;
SET @count = 5;
PRINT 'Count is ' + CAST(@count AS VARCHAR);
```

Code Block

- A group of statements executed together, often enclosed in BEGIN ... END



IF...ELSE



CASE Expression

- Inline conditional logic—used in SELECT, WHERE, ORDER BY, etc.

```
SELECT
    CASE
        WHEN S.GPA >= 2.0 THEN 'Pass'
        ELSE 'Fail'
    END AS Result
FROM Students S;
```

114 %

Results | Messages

	Result
1	Pass
2	Pass
3	Pass

WHILE Loop

- Repeats code block while a condition is true.

```
DECLARE @counter INT = 1;
WHILE @counter <= 5
BEGIN
    PRINT @counter;
    SELECT @counter = @counter + 1;
END
```

114 %

Messages

1
2
3
4
5

BREAK and CONTINUE

- Used within WHILE loops:
    - BREAK exits the loop.
    - CONTINUE skips current iteration and moves to next.

```
DECLARE @counter int
set @counter = 3
WHILE @counter < 10
BEGIN
    SET @counter = @counter + 1;
    IF @counter = 5 CONTINUE;
    IF @counter = 8 BREAK;
    PRINT @counter;
END
```

```
Messages
4
6
7
```

GOTO

- Transfers execution to a labeled part of the script (less recommended for clarity).

```
Declare @x int
set @x = 0
IF @x = 0
    GOTO Skip;
-- Code here is skipped
Skip:
PRINT 'Jumped to Skip label';
```

```
Messages
Jumped to Skip label
```

Stored Procedure (For detail, refer previous manual for SPs)

- Reusable collection of SQL statements with optional input/output parameters.

```
CREATE PROCEDURE GetStudentsByClass @DepartmentID INT
AS
BEGIN
    SELECT * FROM Students S WHERE S.DepartmentID = @DepartmentID;
END
EXEC GetStudentsByClass @DepartmentID = 2;
```

14 %

Results | Messages

	StudentID	StudentName	Email	PhoneNumber	EnrollmentDate	GPA	DepartmentID
1	2	Bob	bob@edu.org	9876543210	2022-09-15	3.60	2

Trigger (For Detail, refer previous manuals specific to triggers)

- Automatically executes in response to a SQL event (INSERT, UPDATE, DELETE).

```
CREATE TRIGGER trg_LogInsert
ON Students
AFTER INSERT
AS
BEGIN
    INSERT INTO AuditLog(Event, EventTime) VALUES ('INSERT', GETDATE());
END
```

Scalar Function

A scalar function is a user-defined or built-in SQL function that returns a single value (a scalar), based on input values or expressions. Unlike table-valued functions (which return rows/columns), scalar functions always return only one value of a defined data type.

Built-In Types

These are provided by SQL and categorized into:

- String Functions: LEN(), UPPER(), LOWER(), SUBSTRING()
- Numeric Functions: ROUND(), ABS(), POWER()
- Date/Time Functions: GETDATE(), DATEADD(), DATEDIFF()
- Conversion Functions: CAST(), CONVERT()
- System Functions: ISNULL(), COALESCE(), NEWID()

User Defined Types

These are custom functions created by users to encapsulate logic and reuse it across queries.

```sql
-- Syntax for UDF Scalar
CREATE FUNCTION FunctionName(@param1 TYPE, @param2 TYPE, ...)
RETURNS ReturnType
AS
BEGIN
    -- Declare variables (optional)
    -- Add logic
    RETURN value;
END
```

Example

```sql
-- Example Scalar UDF
CREATE FUNCTION GetFullName(@first NVARCHAR(50), @last NVARCHAR(50))
RETURNS NVARCHAR(100)
AS
BEGIN
   RETURN @first + ' ' + @last;
END
GO
-- Example Call
SELECT dbo.GetFullName('Ali', 'Khan');
```

14 %

Results | Messages

	(No column name)
1	Ali Khan

Using Scalar Function as Computed Column in SELECT

```sql
SELECT
    StudentID,
    dbo.GetFullName(S.StudentName, ' Khan') AS FullName
FROM Students S;
```

114 %

Results | Messages

	StudentID	FullName
1	1	Alice Khan
2	2	Bob Khan
3	3	Charlie Khan

Using Scalar Function as Computed Column in `ALTER TABLE`

```
ALTER TABLE Students
ADD FullName AS dbo.GetFullName(StudentName, FatherName);

select * from Students
```

114 %

Results Messages

	StudentID	StudentName	Email	PhoneNumber	EnrollmentDate	GPA	DepartmentID	FatherName	FullName
1	1	Alice	alice@edu.org	1234567890	2023-01-10	3.80	1	Khan	Alice Khan
2	2	Bob	bob@edu.org	9876543210	2022-09-15	3.60	2	Khan	Bob Khan
3	3	Charlie	charlie@edu.org	5551234567	2021-06-05	3.90	1	Khan	Charlie Khan

- The column FullName will automatically compute using the function.
- It is virtual (not physically stored unless PERSISTED is specified).

Applications

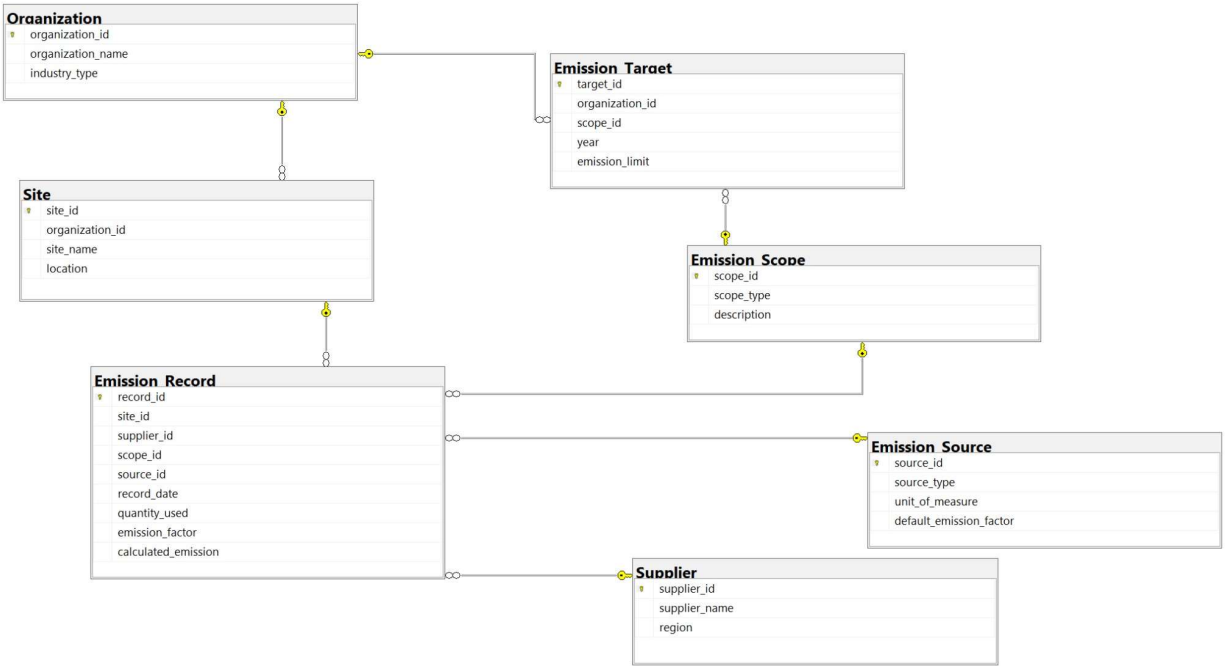
1. Scalar Function
    - Data formatting (e.g., full names, initials)
    - Reusable business logic (e.g., tax calculations)
    - Clean up complex queries
2. Trigger
    - Maintain audit logs
    - Auto-enforce data integrity rules
    - Prevent/limit unwanted data changes
3. Stored Procedure
    - Wrap and reuse large logic blocks
    - Improve performance via precompiled execution
    - Parameterized operations (e.g., filter by user input)

Comparison of Stored Procedures, Triggers and Scalar Functions

Feature	Stored Procedure	Scalar Function	Trigger
Returns	None / Result sets	Single scalar value	No return (executes automatically)
Execution	Manually invoked	Called in expressions	Automatically triggered
Parameters	Supports IN/OUT/INOUT	Input only	None
Can Modify Data	Yes	No	Yes
Usage Context	Batch operations, workflows	Computation, formatting	Auditing, enforcing rules

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

Super Dog and the Carbon Code Crackdown

In a futuristic city called EcoMetra, pollution has been on the rise. But there's hope! The legendary hero, Super Dog, has sniffed out a new mission — to analyze and reduce carbon emissions across the city. With his sharp instincts, high-speed tail-powered jetpack, and a team of brilliant animal sidekicks, he sets out to understand where all the carbon is coming from.

Characters:

- Super Dog – The hero! Can fly, analyze data at super-speed, and communicate with AI systems.
- EcoCat – Expert in Scope 1, 2, and 3 emissions, has infrared vision to detect direct emissions.
- PandaBytes – The coder panda, good with spreadsheets, databases, and modeling.
- Chameleon Camo – Disguises herself to infiltrate polluting facilities and collect raw data.

Mission Objective:

Help Super Dog and his team **gather, calculate, and analyze carbon emissions data** from different **suppliers, sites, and emission scopes** to build a full emissions profile for EcoMetra's organizations.

Write these queries to getting started with in-lab work

1. Write a stored procedure that loops through all organizations and prints the total number of sites per organization using a WHILE loop.
2. Given a source_id, loop through all Emission_Record rows using that source and reset emission_factor = default_emission_factor from Emission_Source.
3. Write a stored procedure that loops month by month (for the past year) and prints total emissions per site_id using calculated_emission.
4. Create an AFTER DELETE trigger on Emission_Record that Inserts the deleted record's key info (record_id, site_id, record_date, calculated_emission) into a log table called Deleted_Emission_Log. (create tables for logging)
5. Add a column last_updated DATETIME (nullable) to Emission_Record and then write an AFTER UPDATE trigger that Sets last_updated = GETDATE() whenever quantity_used or emission_factor is modified.
6. Create a trigger that prevents users from dropping tables in the SuperDogCarbonDB. If a DROP command is issued, cancel it and log the attempt.
7. Log who changed a table's structure and when into Schema_Change_Log(table_name, changed_by, change_time). (Hint: AFTER ALTER_TABLE)

Submission Guidelines

---

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

---

Best of Luck! Happy Querying
