Web Development and Database Administration Level IV

Based on November, 2023 Version-II Curriculum

Module Title: Advanced Structured Query Language

Module code: EIS WDDBA4 M03 1123

Nominal duration: 100 Hours

Prepared by: Ministry of Labor and Skill

November, 2023

Addis Ababa, Ethiopia

Table of Contents

Acknowledgment

Ministry of Labor and Skills wish to extend thanks and appreciation to the many representatives of TVET instructors and respective industry experts who donated their time and expertise to the development of this Teaching, Training and Learning Materials (TTLM).

Acronym

DML…… Data Manipulation Language

RDBMS…… Relational Database Management System

SSMS…… SQL Server Management Studio

Introduction to the Module

In this module, you will explore advanced SQL syntax and functionalities that enable you to perform tasks such as aggregating data, filtering results, joining multiple tables, and handling complex queries. By the end of this module, you will have a comprehensive understanding of advanced SQL techniques, equipping you with the skills to efficiently manipulate and retrieve data from databases and optimize query performance.

This module covers the units:

- Advanced SQL statement
- SQL statements with functions
- SQL statements with aggregation and filtering

This module covers the objectives:

- Construct complex queries involving multiple tables using JOIN statement
- Understand how to use SQL functions to manipulate data, perform calculations, format strings, and extract useful information.
- Understand the GROUP BY clause to group data based on specific criteria
- Filtering data using the HAVING clause

Module Instruction

For effective use this modules trainees are expected to follow the following module instructions

- Read the specific objectives of this Learning Guide.
- Read the information that this module contain.
- Complete the Self-check.
- Submit your accomplished Self-check.
- Do the Operations in the module.
- Do the LAP test in page (if you are ready) and show your output to your teacher.

Unit One: Advanced SQL statements

This unit is developed to provide you the necessary information regarding the following content coverage and topics

- DBMS fundamentals
- Information-based database tools and equipment
- Functions of where clause
- Functions of order by clause
- Boolean operators
- Elimination of duplicated and null values
- Functions of join operator
- Functions of union operator
- Data control language
- Transaction control language

This unit will also assist you to attain the learning outcomes stated in the cover page.

Specifically, upon completion of this learning guide, you will be able to:

- Understand fundamentals of DBMS and identify database tools and equipment
- Retrieve specific columns and sort query output
- Use Boolean operators
- Eliminate duplicated and null values
- Retrieve data from two or more tables
- Combine the result-set of two or more SELECT statements
- Grant and revoke a database
- Understand TCL

1.1. DBMS fundamentals

A database-management system (DBMS) is a collection of interrelated data and a set of programs to access those data. This is a collection of related data with an implicit meaning and

hence is a database. The collection of data, usually referred to as the database, contains information relevant to an enterprise. The primary goal of a DBMS is to provide a way to store and retrieve database information that is both convenient and efficient. By data, we mean known facts that can be recorded and that have implicit meaning. The management system is important because without the existence of some kind of rules and regulations it is not possible to maintain the database. We have to select the particular attributes which should be included in a particular table; the common attributes to create relationship between two tables; if a new record has to be inserted or deleted then which tables should have to be handled etc. These issues must be resolved by having some kind of rules to follow in order to maintain the integrity of the database.

Database systems are designed to manage large bodies of information. Management of data involves both defining structures for storage of information and providing mechanisms for the manipulation of information. In addition, the database system must ensure the safety of the information stored, despite system crashes or attempts at unauthorized access. If data are to be shared among several users, the system must avoid possible anomalous results. So Understanding the fundamentals of DBMS is crucial for anyone working with databases, whether as a database administrator, data analyst, or software developer.

1. Data Organization: One of the primary functions of a DBMS is to organize data in a structured manner. It provides a logical framework, known as a schema, which defines the structure and relationships of the data. The schema includes tables, which consist of rows (records) and columns (attributes). By organizing data into tables, DBMS ensures data integrity, consistency, and easy access.
2. Data Retrieval: DBMS allows users to retrieve data from databases using queries. Queries are written in a language called Structured Query Language (SQL), which is a standard language for interacting with relational databases. With SQL, users can specify the desired data, apply filters, join multiple tables, and sort the results. DBMS optimizes query execution to ensure efficient retrieval of data.
3. Data Manipulation: DBMS enables users to manipulate data by performing various operations such as inserting, updating, and deleting records. These operations ensure data consistency and integrity. DBMS also supports transactions, which are a set of operations

that are executed as a single unit. Transactions ensure that either all operations are successfully completed, or none of them are applied, maintaining data integrity.

4. Data Security: DBMS provides mechanisms to ensure data security and access control. It allows administrators to define user roles and privileges, restricting unauthorized access to sensitive data. DBMS also supports encryption techniques to protect data during transmission and storage.
5. Data Integrity and Consistency: DBMS enforces data integrity rules to maintain the accuracy and consistency of data. It supports constraints such as primary keys, foreign keys, and unique constraints to ensure data integrity. DBMS also provides mechanisms for data validation and error handling.
6. Data Backup and Recovery: DBMS offers features for data backup and recovery to protect against data loss. It allows administrators to schedule regular backups and restore data in case of system failures or disasters. This ensures the availability and reliability of data.

1.2. Database tools and equipment

When it comes to managing and manipulating data effectively, selecting the right database tools and equipment is crucial. The choice of tools and equipment should align with the specific information requirements of your organization or project. This process involves considering various factors such as data volume, complexity, security needs, performance requirements, and scalability.

To begin with, it is essential to assess the data volume and complexity of your project. If you are dealing with large datasets or complex data structures, you may need a robust database management system (DBMS) that can handle the workload efficiently. In such cases, relational database management systems (RDBMS) like SQL, Oracle, or Microsoft SQL Server are commonly used due to their ability to handle structured data and perform complex queries.

Security is another critical aspect to consider when selecting database tools and equipment. If your organization deals with sensitive or confidential data, you should prioritize tools that offer robust security features such as encryption, access controls, and auditing capabilities. Database

systems like PostgreSQL and Microsoft SQL Server provide advanced security features to protect your data from unauthorized access or breaches.

Performance requirements play a significant role in determining the appropriate database tools and equipment. If your application or project demands high-speed data processing and retrieval, you might consider in-memory databases like Redis or Apache Ignite. These databases store data in memory, enabling faster access and query execution compared to traditional disk-based databases.

Scalability is also a crucial factor to consider, especially if your project is expected to grow in terms of data volume or user base. Tools like Apache Cassandra or MongoDB are known for their ability to handle massive amounts of data and provide horizontal scalability, allowing you to add more servers to accommodate increasing demands.

In this module, we will be using Microsoft SQL Server as the database management system for learning and practicing advanced SQL concepts. SQL Server is a popular and widely-used open-source relational database management system that provides a robust and scalable platform for storing, managing, and retrieving data.

SQL Server offers a comprehensive set of features and functionalities that make it an ideal choice for learning advanced SQL. It supports a wide range of SQL commands and syntax, allowing us to explore and practice various advanced techniques effectively. Additionally, SQL Server has excellent performance and reliability, making it suitable for handling large datasets and complex queries. By using Microsoft SQL Server, you will gain hands-on experience with a real-world database management system. You will learn how to set up and configure SQL, create and manage databases, tables, and indexes, and execute SQL queries to manipulate and retrieve data. This practical experience will not only deepen your understanding of advanced SQL concepts but also prepare you for real-world scenarios where SQL is commonly used.

Figure 1.1. SQL

1.3. Functions of where clause

- **Retrieving data from a table**

The SQL **WHERE** clause is used to filter the results obtained by the DML statements such as SELECT, UPDATE and DELETE etc. It is used to extract only those records that fulfill a specified condition. We can retrieve the data from a single table or multiple tables (after join operation) using the WHERE clause.

You can use SQL to **retrieve the columns of a database table** with the SELECT statement. You can retrieve all columns, a single column, or several specific columns. It is then up to your programming language to display that data.

Questions you may have include:

- How do you retrieve all columns?
- How do you retrieve a single column?
- How do you retrieve some columns?

You can write a query to retrieve all the elements in a database table by using the SELECT statement and wildcard (*) indicator.

Select * from table_name;

Note: It is standard practice to add **";"** at the end of your SQL query.

The most common query from a database is to collect or retrieve all the elements in a specific column of the database table.

Select column_name from table_name;

You can also retrieve data from several columns by separating the column names with a comma.

Select column_1, column_2, column_7 from table_name;

Note: Do not place a comma after the last item in the list, because it will result in an error message.

- **WHERE clause with SELECT statement**

Typically, the SELECT statement is used to retrieve data from a table. If we use the WHERE clause with the SELECT statement, we can filter the rows to be retrieved based on a specific condition (or expression).

Example: Assume we have created a table named CUSTOMERS in SQL database using CREATE TABLE statement and inserted some values. The table created is as shown below.

ID	Name	Age	Address	Salary
1	Tilahun	32	Gulele	2000
2	Kebede	25	Arada	1500
3	Chemdesa	23	Lemikura	2000
4	Fulea	25	Kality	6500
5	Kemal	27	Yeka	8500
6	Momona	22	Lafto	4500
7	Tibletse	24	Bole	10000

In the following query, we are fetching the ID, NAME and SALARY fields from the CUSTOMERS table for the records where the SALARY is greater than 2000

- Select ID, Name, Salary from customers where salary > 2000;

ID	Name	Salary
4	Fulea	6500
5	Kemal	8500
6	Momona	4500
7	Tibletse	10000

- **WHERE clause with UPDATE statement**

The UPDATE statement is used to modify the existing records in a table. Using the SQL WHERE clause with the UPDATE statement, we can update particular records. If the WHERE clause is not used, the UPDATE statement would affect all the records of a table. Following is the syntax.

Example: by using the previous customers table, we are incrementing the salary of the customer named Ramesh by 10000 by using the WHERE clause along with the UPDATE statement

- ✓ Update customer set salary= salary + 1000 where Name= 'Tilahun';

ID	Name	Age	Address	Salary
1	Tilahun	32	Gulele	3000

- **Comparison operators in the WHERE clause**

SQL Comparison Operators test whether two given expressions are the same or not. These operators are used in SQL conditional statements while comparing one expression with another and they return a Boolean value which can be either TRUE or FALSE. The result of an SQL comparison operation can be UNKNOWN when one or another operand has its value as NULL.

Here is a list of all the comparison operators available in SQL.

Operator	Description
=	Equal to
!=	Not equal
<>	Not equal

>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
!<	Not less than
!>	Not greater than

Example: by using the previous customers table, we can write a query with a comparison operator

- Select * from customers where age != 25;

ID	Name	Age	Address	Salary
1	Tilahun	32	Gulele	2000
3	Chemdesa	23	Lemikura	2000
5	Kemal	27	Yeka	8500
6	Momona	22	Lafto	4500
7	Tibletse	24	Bole	10000

- Select * from customers where age >= 25;

ID	Name	Age	Address	Salary
1	Tilahun	32	Gulele	2000
2	Kebede	25	Arada	1500
4	Fulea	25	Kality	6500
5	Kemal	27	Yeka	8500

- **CREATE INDEX Statement**

The CREATE INDEX statement is used to create indexes in tables. They are used to retrieve data from the database more quickly than otherwise. The users cannot see the indexes, they are just used to speed up searches/queries.

Note: Updating a table with indexes takes more time than updating a table without (because the indexes also need an update). So, only create indexes on columns that will be frequently searched against.

- ✓ CREATE INDEX Syntax

Creates an index on a table. Duplicate values are allowed:

- CREATE INDEX index_name
  ON table_name (column1, column2, ...);
- CREATE INDEX index_name
  ON table_name (column1, column2, ...) ASC or DES;

1.4. Functions of order by clause

The SQL **ORDER BY** clause is used to sort the data in either ascending or descending order, based on one or more columns. This clause can sort data by a single column or by multiple columns. Sorting by multiple columns can be helpful when you need to sort data hierarchically, such as sorting by state, city, and then by the person's name.

ORDER BY is used with the SQL SELECT statement and is usually specified after the WHERE, HAVING, and GROUP BY clauses.

Following are the important points about ORDER BY Clause

- To sort the data in ascending order, we use the keyword **ASC**.
- To sort the data in descending order, we use the keyword **DESC**.

In addition to sorting records in ascending order or descending order, the ORDER BY clause can also sort the data in a database table in a preferred order.

Example: by using the previous product table

Sort the products from highest to lowest price

- SELECT * FROM Products
  ORDER BY Price DESC;

Sort the products from lowest to highest price

- SELECT * FROM Products
  ORDER BY Price ASC;

Sort the products alphabetically by using Product name

- SELECT * FROM Products
  ORDER BY Product_Name;

Sort the products by Product Name in reverse order

- SELECT * FROM Products
  ORDER BY Product_Name DESC;

1.5. Boolean operators

Boolean operators in SQL are logical operators used to combine or manipulate conditions in a query.

Operator	Description
ALL	TRUE if all of the sub query values meet the condition
AND	TRUE if all the condition separated by AND is true
ANY	TRUE if any of the sub query values meet the condition
BETWEEN	TRUE if the operand is within the range of comparisons
EXISTS	TRUE if the sub query returns one or more records
IN	TRUE if the operand is equal to one of a list of expressions
LIKE	TRUE if the operand matches a pattern
NOT	Displays a record if the condition(s) is NOT TRUE
OR	TRUE if any of the conditions separated by OR is true

Example: Assume we have created a table named CUSTOMERS in SQL database using CREATE TABLE statement and inserted some values. The table created is as shown below.

ID	Name	Age	Address	Salary
1	Tilahun	32	Gulele	2000
2	Kebede	25	Arada	1500
3	Chemdesa	23	Lemikura	2000
4	Fulea	25	Kality	6500
5	Kemal	27	Yeka	8500
6	Momona	22	Lafto	4500
7	Tibletse	24	Bole	10000

- **SQL ALL Operator**

The ALL operator:

- Returns a Boolean value as a result
- Returns TRUE if ALL of the sub query values meet the condition
- It is used with SELECT, WHERE and HAVING statements

ALL means that the condition will be true only if the operation is true for all values in the range.

ALL operator syntax

- SELECT ALL *column_name(s)*
  FROM *table_name*
  WHERE *condition*;

- **WHERE clause with AND, OR operators**

We can use AND and OR operators together in SQL to combine multiple conditions in a WHERE clause to filter rows that meets the specified criteria. The AND operator will make sure only those rows are filtered that satisfy all the conditions and the OR operator will filter records that satisfy any one of the specified conditions. However, this is only used when specifying one condition is not enough to filter all the required rows.

Following is the syntax for using the AND and OR operators in a WHERE clause

- ✓ WHERE (condition1 OR condition2) AND condition3;

Example: by using the previous customers table, we are retrieving all rows from the CUSTOMERS table based on some conditions. The parentheses control the order of evaluation so that the OR operator is applied first, followed by the AND operator.

- ➢ Select * from customers where (age = 25 OR salary < 4500) AND (name = 'Chemdesa' OR name = 'Momona');

ID	Name	Age	Address	Salary
3	Chemdesa	23	Lemikura	2000

- **SQL ANY Operator**

The ANY operator:

- ➢ Returns a Boolean value as a result
- ➢ Returns TRUE if ANY of the sub query values meet the condition

ANY means that the condition will be true if the operation is true for any of the values in the range.

ANY operator syntax

- ✓ SELECT *column_name(s)*
  FROM *table_name*
  WHERE *column_name operator* ANY
  (SELECT *column_name*
  FROM *table_name*
  WHERE *condition*);

Note: The operator must be a standard comparison operator (=, <>, !=, >, >=, <, or <=).

Example: in the following example if we want to retrieve name of the customers whose salary is 2000, we can write this query

- ✓ SELECT Name
  FROM customers
  WHERE ID = ANY
  (SELECT ID
  FROM customers
  WHERE salary=2000;

Name
Tilahun
Chemdesa

- **SQL BETWEEN Operator**

The **BETWEEN** operator is a logical operator in SQL, that is used to retrieve the data within a specified range. The retrieved values can be integers, characters, or dates. You can use the **BETWEEN** operator to replace a combination of "greater than equal AND less than equal" conditions.

- ➢ SELECT column_name(s)
  FROM table_name
  WHERE column_name BETWEEN value1 AND value2;

Example: by using the previous customers table, we can write a query that will display all customers whose age is between 20 and 30.

- ➢ Select * from customers where age between 20 and 30;

ID	Name	Age	Address	Salary
2	Kebede	25	Arada	1500
3	Chemdesa	23	Lemikura	2000
4	Fulea	25	Kality	6500
5	Kemal	27	Yeka	8500
6	Momona	22	Lafto	4500
7	Tibletse	24	Bole	10000

Example: by using the previous customers table, we can write a query that will display all customers whose name starts between the a and g.

- Select * from customers where name between ‘a’ and ‘g’;

ID	Name	Age	Address	Salary
3	Chemdesa	23	Lemikura	2000
4	Fulea	25	Kality	6500

- ✓ **NOT BETWEEN Operator**

The **NOT BETWEEN** operator in SQL works in exactly opposite to the BETWEEN operator. This is used to retrieve the data which is not present in the specified range.

Example: Select * from customers where age not between 20 and 30;

ID	Name	Age	Address	Salary
1	Tilahun	32	Gulele	2000

- **SQL EXISTS Operator**
  - The EXISTS operator is used to test for the existence of any record in a sub query. It returns TRUE if the sub query returns one or more records.

EXISTS operator Syntax

- SELECT *column_name(s)*
  FROM *table_name*
  WHERE EXISTS
  (SELECT *column_name* FROM *table_name* WHERE *condition*);

- **WHERE clause with IN operator**

Using the IN operator you can specify the list of values or sub query in the where clause. If you use WHERE and IN with the SELECT statement, it allows us to retrieve the rows in a table that match any of the values in the specified list. Following is the syntax for it

Example: by using the previous customers table, suppose you want to display records with NAME values Khilan, Hardik and Muffy from the CUSTOMERS table, you can use the following query

- Select * from customers where name in (‘Tilahun', 'Chemdesa', 'momona');

ID	Name	Age	Address	Salary
1	Tilahun	32	Gulele	3000
3	Chemdesa	23	Lemikura	2000
6	Momona	22	Lafto	4500

- **WHERE clause with NOT IN operator**

The WHERE clause with NOT IN operator is the negation of WHERE clause with the IN operator.

- ✓ If you use WHERE with the IN operator, the DML statement will act on the list of values (of a column) specified
- ✓ Whereas, if you use WHERE with the NOT IN operator, the DML operation is performed on the values (of a column) that are not there in the specified list.

Example: by using the previous customers table, we are displaying the records from CUSTOMERS table, where AGE is NOT equal to **25**, **23** and **22**.

- ✓ Select * from customers where age NOT IN (25, 23, 22);

ID	Name	Age	Address	Salary
1	Tilahun	32	Gulele	2000
5	Kemal	27	Yeka	8500
7	Tibletse	24	Bole	10000

- **SQL LIKE Operator**

- The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.
- There are two wildcards often used in conjunction with the LIKE operator:
  - ✓ The percent sign % represents zero, one, or multiple characters
  - ✓ The underscore sign _ represents one, single character

LIKE operator syntax

- SELECT column1, column2, ...
  FROM table_name
  WHERE columnN LIKE pattern;

Example: by using the previous customers table, we can write a query which would display all the records where the name starts with K and is at least 3 characters in length

- ✓ Select * from customers where name LIKE 'K__%';

ID	Name	Age	Address	Salary
2	Kebede	25	Arada	1500
5	Kemal	27	Yeka	8500

- **SQL NOT Operator**
  - The NOT operator is used in combination with other operators to give the opposite result, also called the negative result.

NOT operator syntax

- SELECT column1, column2, ...
  FROM table_name
  WHERE NOT condition;

1.6. Elimination of duplicated and null values

- **Duplicated values**

Duplicates can be a big problem in SQL databases as they can slow down query performance and waste valuable storage space.

When the result set from a SELECT statement contains duplicate rows, you may want to remove them and keep every row data to be unique for a column or combination of columns. You can use the DISTINCT identifier to eliminate duplicate records from the result set.

Consider the following facts when using DISTINCT identifier in a SELECT statement:

- In a SELECT statement, include DISTINCT keyword after the SELECT clause.
- Multiple columns can be specified after DISTINCT keyword. In this case, the result set contains distinct combination of data from these columns.

DISTINCT keyword syntax

- SELECT DISTINCT column1, column2, ...
  FROM table_name;

- **Null Value**

A null value indicates no value. It means that the column value is absent in a row. A null value is not the same as a blank space or a zero value. A zero value is an integer and a blank space is a character while a null value is the one that has been left blank.

To exclude the null values from a table we need to use IS NOT NULL operator with the WHERE clause. **IS NOT NULL Operator** is used to test for non-empty values.

IS NOT NULL operator syntax:

- SELECT column_names
  FROM table_name WHERE column_name IS NOT NULL;

1.7. Functions of join operator

A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

The INNER JOIN keyword selects records that have matching values in both tables. It is the most common type of join. Inner joins combine records from two tables whenever there are matching values in a field common to both tables.

Inner Join clause in SQL Server creates a new table (not physical) by combining rows that have matching values in two or more tables. This join is based on a logical relationship (or a common field) between the tables and is used to retrieve data that appears in both tables.

INNER JOIN operator syntax

```
SELECT column_name(s)
FROM table1
INNER JOIN table2
ON table1.column_name = table2.column_name;
```

- **SQL LEFT JOIN Keyword**

The LEFT JOIN keyword returns all records from the left table (table1), and the matching records from the right table (table2). The result is 0 records from the right side, if there is no match.

LEFT JOIN keyword syntax

```
SELECT column_name(s)
FROM table1
LEFT JOIN table2
ON table1.column_name = table2.column_name;
```

Note: In some databases LEFT JOIN is called LEFT OUTER JOIN.

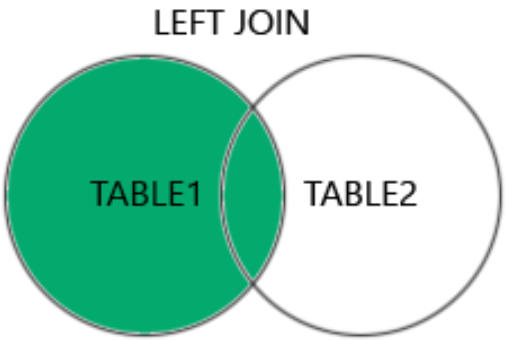


Figure 1.1. Left Join

- **SQL RIGHT JOIN Keyword**

The RIGHT JOIN keyword returns all records from the right table (table2), and the matching records from the left table (table1). The result is 0 records from the left side, if there is no match.

RIGHT JOIN Syntax

```
SELECT column_name(s)
FROM table1
RIGHT JOIN table2
ON table1.column_name = table2.column_name;
```

Note: In some databases RIGHT JOIN is called RIGHT OUTER JOIN.

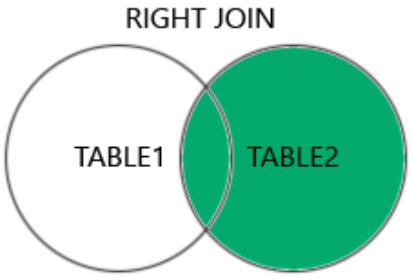


Figure 1.2. Right Join

- **SQL FULL OUTER JOIN Keyword**

The FULL OUTER JOIN keyword returns all records when there is a match in left (table1) or right (table2) table records.

Tip: FULL OUTER JOIN and FULL JOIN are the same.

FULL OUTER JOIN Syntax

- SELECT column_name(s)
  FROM table1 FULL OUTER JOIN table2
  ON table1.column_name = table2.column_name WHERE condition;

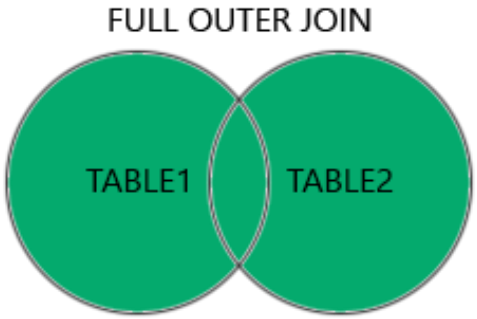


Figure 1.3. Full outer Join

Note: FULL OUTER JOIN can potentially return very large result-sets!

1.8. Functions of union operator

The UNION operator is used to combine the result-set of two or more SELECT statements.

Every SELECT statement within UNION must have the same number of columns

The columns must also have similar data types

The columns in every SELECT statement must also be in the same order

UNION operator syntax

```
SELECT column_name(s) FROM table1
UNION
SELECT column_name(s) FROM table2;
```

UNION operator with where clause syntax

```
SELECT column_name(s) FROM table1 WHERE condition1
UNION SELECT column_name(s) FROM table2 WHERE condition2
```

1.9. Data control language

DCL stands for Data Control Language in Structured Query Language (SQL). As the name suggests these commands are used to control privilege in the database. The privileges (Right to access the data) are required for performing all the database operations like creating tables, views, or sequences.

DCL command is a statement that is used to perform the work related to the rights, permissions, and other control of the database system.

Need Of DCL commands

- Unauthorized access to the data should be prevented in order to achieve security in our database
- DCL commands maintain the database effectively than anyone else other than database administrator is not allowed to access the data without permission.
- These commands provide flexibility to the data administrator to set and remove database permissions in granular fashion.

GRANT

This command is used to grant permission to the user to perform a particular operation on a particular object. If you are a database administrator and you want to restrict user accessibility such as one who only views the data or may only update the data. You can give the privilege permission to the users according to your wish.

Syntax:

GRANT SELECT, UPDATE ON MY_TABLE TO SOME_USER, ANOTHER_USER;

REVOKE

This command is used to take permission/access back from the user. If you want to return permission from the database that you have granted to the users at that time you need to run REVOKE command.

Syntax:

REVOKE SELECT, UPDATE ON MY_TABLE FROM USER1, USER2;

1.10. Transaction control language

Transactions group a set of tasks into a single execution unit. Each transaction begins with a specific job and ends when all the tasks in the group successfully completed. If any of the tasks fail, the transaction fails. Therefore, a transaction has only two results: success or failure.
BEGIN:

Incomplete steps result in the failure of the transaction. A database transaction, by definition, must be atomic, consistent, isolated, and durable.
These are popularly known as ACID properties. These properties can ensure the concurrent execution of multiple transactions without conflict.

Properties of Transaction

- Atomicity: The outcome of a transaction can either be completely successful or completely unsuccessful. The whole transaction must be rolled back if one part of it fails.
- Consistency: Transactions maintain integrity restrictions by moving the database from one valid state to another.
- Isolation: Concurrent transactions are isolated from one another, assuring the accuracy of the data.
- Durability: Once a transaction is committed, its modifications remain in effect even in the event of a system failure.

1. COMMIT

   This command is used to save the data permanently.

   Whenever we perform any of the DML command like -INSERT, DELETE or UPDATE, these can be rollback if the data is not stored permanently. So, in order to be at the safer side COMMIT command is used.

- Syntax: commit;

2. ROLLBACK

   This command is used to get the data or restore the data to the last savepoint or last committed state. If due to some reasons the data inserted, deleted or updated is not correct, you can roll back the data to a particular savepoint or if savepoint is not done, then to the last committed state.

   - Syntax: rollback;

3. SAVEPOINT

   This command is used to save the data at a particular point temporarily, so that whenever needed can be rollback to that particular point.

   - Syntax: Savepoint A;

Example: Consider the following student table

Name	Marks
John	79
Jolly	65
Shuzan	70

- UPDATE STUDENT
  SET NAME = 'Sherlock'
  WHERE NAME = 'Jolly';
  COMMIT;
  ROLLBACK;

By using the above command, you can update the record and save it permanently by using COMMIT command.

Now after COMMIT:

Name	Marks
John	79
Sherlock	65
Shuzan	70

If commit was not performed then the changes made by the update command can be rollback.

Now if no COMMIT is performed.

- UPDATE STUDENT
  SET NAME = ‘Sherlock’
  WHERE STUDENT_NAME = ‘Jolly’;

After update command the table will be:

Name	Marks
John	79
Sherlock	65
Shuzan	70

Now if ROLLBACK is performed on the above table:

rollback;

After Rollback:

Name	Marks
John	79
Jolly	65
Shuzan	70

If on the above table savepoint is performed:

- INSERT into STUDENT
  VALUES ('Jack', 95);
  Commit;
- UPDATE NAME
  SET NAME= ‘Rossie’
  WHERE marks= 70;
  SAVEPOINT A;
- INSERT INTO STUDENT
  VALUES (‘Zack’, 76);
  SAVEPOINT B;
- INSERT INTO STUDENT
  VALUES (‘Bruno’, 85);
  SAVEPOINT C;
- SELECT * FROM STUDENT;

Name	Marks
John	79
Jolly	65
Rossie	70
Jack	95
Zack	76
Bruno	85

Now if we Rollback to SAVEPOINT B:

Rollback to B;

The resulting Table will be-

Name	Marks
John	79
Jolly	65
Rossie	70
Jack	95
Zack	76

Now if we Rollback to SAVEPOINT A:

Rollback to A;

The resulting Table will be-

Name	Marks
John	79
Jolly	65
Rossie	70
Jack	95

Self-Check 1

Part I: Say true or false

_____ 1. The inner join technique is useful when data from several relations are to be retrieved and displayed and the relationships are not necessarily nested.

_____ 2. The UNION clause is used to combine the output from multiple queries together into a single result table.

_____ 3. With the UNION clause, each query involved must output the same number of columns, and they must be UNION compatible.

_____ 4. Difference between WHERE and HAVING clause is that the WHERE clause allows you to filter data from a group of rows.

_____ 5. You can't use aggregate functions with order by clause.

_____ 6. Both group by and order by clauses serve similar function with similar purpose.

Part II: Choose the best answer

_____ 1. Which of the following is NOT a fundamental concept of a Database Management System (DBMS)?

a) Data modeling

b) Data storage

c) Data visualization

d) Data retrieval

_____ 2. Which of the following is an example of a database tool used for data manipulation?

a) SQL Server Management Studio (SSMS)

b) Microsoft Excel

c) Notepad

d) Google Chrome

_____ 3. What is the primary function of the WHERE clause in SQL?

a) To eliminate duplicated values

b) To perform mathematical calculations

c) To specify the order of the result set

d) To filter rows based on specific conditions

_____ 4. What is the purpose of the ORDER BY clause in SQL?

a) To combine multiple tables into a single result set

b) To eliminate null values from the result set

c) To sort the result set based on specified columns

d) To perform arithmetic operations on the result set

_____ 5. Which of the following is a Boolean operator used in SQL?

a) AND c) BETWEEN

b) NOT NULL d) LIKE

_____ 6. TCL statements are primarily used for

a) Defining table structures

b) Specifying conditions in queries

c) Controlling transaction behavior

d) Committing or rolling back changes

e) Creating new databases

_____ 7. Which statements are examples of Data Control Language (DCL) statements?

a) SELECT

b) INSERT

c) GRANT

d) UPDATE

e) DELETE

Operation Sheet-1.1 Create database

Operation title: Create a Merchant database

Purpose: To create a merchant database with customer, supplier, customer_order, orderItem and product table their respective relationships

Equipment tools and materials: SQL server 2012

Step 1: Create database merchant;

Step 2: create customer table

In order to create customer table, we will write the following SQL statement

- CREATE TABLE Customers (

```
Id          int          identity primary key,
FirstName   nvarchar(40)   not null,
LastName    nvarchar(40)   not null,
City        nvarchar(40)   null,
Country     nvarchar(40)   null,
Phone       nvarchar(20)   null)
```

Step 3: Insert into customers table

- INSERT INTO [Customers] ([Id],[FirstName],[LastName],[City],[Country],[Phone])VALUES(1,'Maria','Anders','Berlin','Germany','')

- INSERT INTO [Customers] ([Id],[FirstName],[LastName],[City],[Country],[Phone])VALUES(2,'Ana','Trujillo','México D.F.','Mexico','(5) 555-4729')
- INSERT INTO [Customers] ([Id],[FirstName],[LastName],[City],[Country],[Phone])VALUES(3,'Antonio','Moreno','México D.F.','Mexico','(5) 555-3932')
- INSERT INTO [Customers] ([Id],[FirstName],[LastName],[City],[Country],[Phone])VALUES(4,'Thomas','Hardy','London','UK','(171) 555-7788')
- INSERT INTO [Customers] ([Id],[FirstName],[LastName],[City],[Country],[Phone])VALUES(5,'Christina','Berglund','Luleå','Sweden','0921-12 34 65')
- INSERT INTO [Customers] ([Id],[FirstName],[LastName],[City],[Country],[Phone])VALUES(6,'Hanna','Moos','Mannheim','Germany','0621-08460')
- INSERT INTO [Customers] ([Id],[FirstName],[LastName],[City],[Country],[Phone])VALUES(7,'Frédérique','Citeaux','Strasbourg','France','88.60.15.31')
- INSERT INTO [Customers] ([Id],[FirstName],[LastName],[City],[Country],[Phone])VALUES(8,'Martín','Sommer','Madrid','Spain','(91) 555 22 82')
- INSERT INTO [Customers] ([Id],[FirstName],[LastName],[City],[Country],[Phone])VALUES(9,'Laurence','Lebihan','Marseille','France','91.24.45.40')
- INSERT INTO [Customers] ([Id],[FirstName],[LastName],[City],[Country],[Phone])VALUES(10,'Elizabeth','Lincoln','Tsawassen','Canada','(604) 555-4729')

Step 4: To retrieve the customers table, we can write the following SQL statement

- Select * from customers

Quality Criteria: your output should look like this

Results Messages

Id	FirstName	LastName	City	Country	Phone
1	Maria	Anders	Berlin	Germany	030-0074321
2	Ana	Trujillo	México D.F.	Mexico	(5) 555-4729
3	Antonio	Moreno	México D.F.	Mexico	(5) 555-3932
4	Thomas	Hardy	London	UK	(171) 555-7788
5	Christina	Berglund	Luleå	Sweden	0921-12 34 65
6	Hanna	Moos	Mannheim	Germany	0621-08460
7	Frédérique	Citeaux	Strasbourg	France	88.60.15.31
8	Martín	Sommer	Madrid	Spain	(91) 555 22 82
9	Laurence	Lebihan	Marseille	France	91.24.45.40
10	Elizabeth	Lincoln	Tsawassen	Canada	(604) 555-4729

Step 5: Here we will create a supplier table (refer step 2**)**

Step 6: Insert values in to supplier table (refer step 3**)**

Step 7: To retrieve the supplier table, we can write the following SQL statement

- Select * from supplier

Id	CompanyName	ContactName	City	Country	Phone	Fax
1	Exotic Liquids	Charlotte Cooper	London	UK	(171) 555-2222	NULL
2	New Orleans Cajun Delights	Shelley Burke	New Orleans	USA	(100) 555-4822	NULL
3	Grandma Kelly's Homestead	Regina Murphy	Ann Arbor	USA	(313) 555-5735	(313) 555-3349
4	Tokyo Traders	Yoshi Nagase	Tokyo	Japan	(03) 3555-5011	NULL
5	Cooperativa de Quesos 'Las Cabras'	Antonio del Valle Saavedra	Oviedo	Spain	(98) 598 76 54	NULL
6	Mayumi's	Mayumi Ohno	Osaka	Japan	(06) 431-7877	NULL
7	Pavlova, Ltd.	Ian Devling	Melbourne	Australia	(03) 444-2343	(03) 444-6588
8	Specialty Biscuits, Ltd.	Peter Wilson	Manchester	UK	(161) 555-4448	NULL
9	PB Knäckebröd AB	Lars Peterson	Göteborg	Sweden	031-987 65 43	031-987 65 91
10	Refrescos Americanas LTDA	Carlos Diaz	Sao Paulo	Brazil	(11) 555 4640	NULL

Step 8: Here we will create Customer_order table

In order to create the customer_order table, we will write the following SQL statement

- CREATE TABLE "customer_Order" (

  Id int identity primary key,

```
OrderDate          datetime        not null default getdate(),
OrderNumber        nvarchar(10)    null,
CustomerId         int             not null foreign key references
customers (id)
TotalAmount        decimal(12,2)   null default 0,)
Constraint PK_ORDER primary key (Id)
)
```

Step 9: Insert values in to customer_order table (refer step 3**)**

Step 10: To retrieve the Customer_order table, we can write the following SQL statement

- Select * from Customer_order

Quality Criteria: your output should look like this

Id	OrderDate	OrderNumber	CustomerId	TotalAmount
1	2012-07-04 00:00:00.000	542378	8	440.00
2	2012-07-05 00:00:00.000	542379	7	1863.40
3	2012-07-08 00:00:00.000	542380	4	1813.00
4	2012-07-08 00:00:00.000	542381	3	670.80
5	2012-07-09 00:00:00.000	542382	6	3730.00
6	2012-07-10 00:00:00.000	542383	1	1444.80
7	2012-07-11 00:00:00.000	542384	4	625.20
8	2012-07-12 00:00:00.000	542385	2	2490.50
9	2012-07-15 00:00:00.000	542386	9	517.80
10	2012-07-16 00:00:00.000	542387	10	1119.90

Step 11: Here we will create OrderItem table (refer step 8**)**

Step 12: Insert values in to OrderItem table (refer step 3)

Step 13: To retrieve the OrderItem table, we can write the following SQL statement

- Select * from OrderItem

Quality Criteria: your output should look like this

Id	OrderId	ProductId	UnitPrice	Quantity
1	1	1	14.00	12
2	1	4	9.80	10
3	1	2	34.80	5
4	2	7	18.60	9
5	2	5	42.40	40
6	3	8	7.70	10
7	3	9	42.40	35
8	3	6	16.80	15
9	4	3	16.80	6
10	4	10	15.60	15

Step 14: Create Product table (refer step 8)

Step 15: Insert values in to product table (refer step 3**)**

Step 16: To retrieve the Product table, we can write the following SQL statement

- Select * from Product

Quality Criteria: your output should look like this

Id	ProductName	SupplierId	UnitPrice	Package	IsDiscontinued
1	Chai	1	18.00	10 boxes x 20 bags	0
2	Chang	1	19.00	24 - 12 oz bottles	0
3	Aniseed Syrup	1	10.00	12 - 550 ml bottles	0
4	Chef Anton's Cajun Seasoning	2	22.00	48 - 6 oz jars	0
5	Chef Anton's Gumbo Mix	2	21.35	36 boxes	1
6	Grandma's Boysenberry Spread	3	25.00	12 - 8 oz jars	0
7	Uncle Bob's Organic Dried Pears	3	30.00	12 - 1 lb pkgs.	0
8	Northwoods Cranberry Sauce	3	40.00	12 - 12 oz jars	0
9	Mishi Kobe Niku	4	97.00	18 - 500 g pkgs.	1
10	Ikura	4	31.00	12 - 200 ml jars	0

Step 17: Create index for the merchant database tables

- create index IndexCustomerName on Customers (LastName, FirstName)
- create index IndexSupplierCountry on Supplier (Country ASC)
- create index IndexProductSupplierId on Product (SupplierId ASC)
- create index IndexOrderItemProductId on OrderItem (ProductId ASC)
- create index IndexCustomerorderId on "customer_Order" (CustomerId ASC)

Operation Sheet-1.2 where clause with comparison operators

Operation title: where clause with comparison operators

Purpose: To show functionalities of where clause with comparison operators

Equipment tools and materials: SQL server 2012

Step 1: Use the above merchant database operation sheet 1.1

Step 2: here we want to retrieve customers whose country is Mexico, so we can write the following query

- Select Id, FirstName, LastName, City, Country, Phone
  FROM Customers
  WHERE Country = 'Mexico'

Quality Criteria: your output should look like this

Results Messages

Id	FirstName	LastName	City	Country	Phone
2	Ana	Trujillo	México D.F.	Mexico	(5) 555-4729
3	Antonio	Moreno	México D.F.	Mexico	(5) 555-3932

Step 3: here we want retrieve orders whose unit price is greater than 30, so we can write the following SQL statement

- Select * from OrderItem where unitprice>30;

Quality Criteria: your output should look like this

Id	OrderId	ProductId	UnitPrice	Quantity
3	1	2	34.80	5
5	2	5	42.40	40
7	3	9	42.40	35

Step 4: here we want retrieve to list products with order quantities greater than or equal to 15, so we can write the following SQL statement

- SELECT ProductName
  FROM Product
  WHERE Id IN (SELECT ProductId
  FROM OrderItem
  WHERE Quantity >= 15)

Quality Criteria: your output should look like this

ProductName
Chef Anton's Gumbo Mix
Grandma's Boysenberry Spread
Mishi Kobe Niku
Ikura

Operation Sheet-1.3 where and order by clause

Operation title: where and order by clause

Purpose: To show functionalities of where and order by clause

Equipment tools and materials: SQL server 2012

Step 1: Use the above merchant database from operation sheet 1.1

Step 2: here we want to retrieve 50% of the customers record, so we can write the following SQL statement

- Select top 5 * from customers where Country='mexico'

Quality Criteria: your output should look like this

Id	FirstName	LastName	City	Country	Phone
2	Ana	Trujillo	México D.F.	Mexico	(5) 555-4729
3	Antonio	Moreno	México D.F.	Mexico	(5) 555-3932

Step 3: here we want to list all suppliers with the number of products they offer, so we can write the following SQL statement

- SELECT CompanyName,
  ProductCount = (SELECT COUNT(P.id)
  FROM [Product] P
  WHERE P.SupplierId = S.Id)
  FROM Supplier S order by companyname DESC

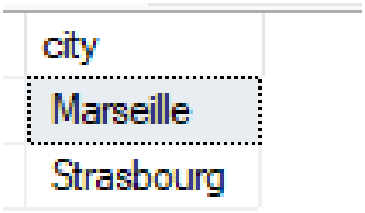
Quality Criteria: your output should look like this

CompanyName	ProductCount
Tokyo Traders	2
Specialty Biscuits, Ltd.	0
Refrescos Americanas LTDA	0
PB Knäckebröd AB	0
Pavlova, Ltd.	0
New Orleans Cajun Delights	2
Mayumi's	0
Grandma Kelly's Homestead	3
Exotic Liquids	3
Cooperativa de Quesos 'Las Cabras'	0

Step 4: here we want to list all French customer cities (without duplicates)

- SELECT distinct city from customers where Country = 'france';

Quality Criteria: your output should look like this



city
Marseille
Strasbourg

Step 5: here we want to list all suppliers that have no fax, we can write the following query

- SELECT Id, CompanyName, Phone, Fax
  FROM Supplier
  WHERE Fax IS NULL

Quality Criteria: your output should look like this

Id	CompanyName	Phone	Fax
1	Exotic Liquids	(171) 555-2222	NULL
2	New Orleans Cajun Delights	(100) 555-4822	NULL
4	Tokyo Traders	(03) 3555-5011	NULL
5	Cooperativa de Quesos 'Las Cabras'	(98) 598 76 54	NULL
6	Mayumi's	(06) 431-7877	NULL
8	Specialty Biscuits, Ltd.	(161) 555-4448	NULL
10	Refrescos Americanas LTDA	(11) 555 4640	NULL

Operation Sheet-1.4 Working with Boolean operators

Operation title: Working with Boolean Operators

Purpose: To show functionalities of LIKE, EXISTS and IN operators

Equipment tools and materials: SQL server 2012

Step 1: Use the above merchant database from operation sheet 1.1

Step 2: Here we want to list all products that are packaged in jars, we can write the following query

- SELECT *
  FROM Product
  WHERE Package LIKE '%jars%'

Quality Criteria: your output should look like this

Id	ProductName	SupplierId	UnitPrice	Package	IsDiscontinued
4	Chef Anton's Cajun Seasoning	2	22.00	48 - 6 oz jars	0
6	Grandma's Boysenberry Spread	3	25.00	12 - 8 oz jars	0
8	Northwoods Cranberry Sauce	3	40.00	12 - 12 oz jars	0
10	Ikura	4	31.00	12 - 200 ml jars	0

Step 3: here we want to list customers with orders over $2000, we can write the following query

- SELECT *
  FROM Customers
  WHERE EXISTS
  (SELECT Id
  FROM [customer_Order]
  WHERE CustomerId = Customers.Id
  AND TotalAmount > 2000)

Quality Criteria: your output should look like this

Id	FirstName	LastName	City	Country	Phone
2	Ana	Trujillo	México D.F.	Mexico	(5) 555-4729
6	Hanna	Moos	Mannheim	Germany	0621-08460

Step 4: here we want to list customers who are from London or Paris, we can write the following query

- SELECT firstname
  FROM Customers
  WHERE City IN ('Paris','London')

Quality Criteria: your output should look like this

firstname
Thomas

Operation Sheet-1.5 Working with Join

Operation title: Working with Join

Purpose: To show functionalities of Join, left join, right join and full join

Equipment tools and materials: SQL server 2012

Step 1: Use the above merchant database from operation sheet 1.1

Step 2: To List all suppliers with their products we can write the following query

- SELECT CompanyName, ProductName
  FROM Supplier S
  JOIN Product P ON S.Id = P.SupplierId

Quality Criteria: your output should look like this

CompanyName	ProductName
Exotic Liquids	Chai
Exotic Liquids	Chang
Exotic Liquids	Aniseed Syrup
New Orleans Cajun Delights	Chef Anton's Cajun Seasoning
New Orleans Cajun Delights	Chef Anton's Gumbo Mix
Grandma Kelly's Homestead	Grandma's Boysenberry Spread
Grandma Kelly's Homestead	Uncle Bob's Organic Dried Pears
Grandma Kelly's Homestead	Northwoods Cranberry Sauce
Tokyo Traders	Mishi Kobe Niku
Tokyo Traders	Ikura

Step 3: To list all suppliers and their products, including suppliers with no products we can write the following query

- SELECT CompanyName, ProductName

FROM Supplier S

LEFT JOIN Product P ON S.Id = P.SupplierId

Quality Criteria: your output should look like this

CompanyName	ProductName
Exotic Liquids	Chai
Exotic Liquids	Chang
Exotic Liquids	Aniseed Syrup
New Orleans Cajun Delights	Chef Anton's Cajun Seasoning
New Orleans Cajun Delights	Chef Anton's Gumbo Mix
Grandma Kelly's Homestead	Grandma's Boysenberry Spread
Grandma Kelly's Homestead	Uncle Bob's Organic Dried Pears
Grandma Kelly's Homestead	Northwoods Cranberry Sauce
Tokyo Traders	Mishi Kobe Niku
Tokyo Traders	Ikura
Cooperativa de Quesos 'Las Cabras'	NULL
Mayumi's	NULL
Pavlova, Ltd.	NULL
Specialty Biscuits, Ltd.	NULL
PB Knäckebröd AB	NULL
Refrescos Americanas LTDA	NULL

Step 4: To list customers that have not placed orders we can write the following query

- SELECT FirstName, LastName, City, Country, TotalAmount
  FROM [customer_Order] O
  RIGHT JOIN Customers C ON O.CustomerId = C.Id
  WHERE TotalAmount IS NULL

Quality Criteria: your output should look like this

FirstName	LastName	City	Country	TotalAmount
Christina	Berglund	Luleå	Sweden	NULL

Step 5: To match all customers and suppliers by country we can write the following query

- SELECT C.FirstName, C.LastName, C.Country AS CustomerCountry,
  S.Country AS SupplierCountry, S.CompanyName
  FROM Customers C
  FULL JOIN Supplier S ON C.Country = S.Country
  ORDER BY C.Country, S.Country

Quality Criteria: your output should look like this

FirstName	LastName	CustomerCountry	SupplierCountry	CompanyName
NULL	NULL	NULL	Brazil	Refrescos Americanas LTDA
NULL	NULL	NULL	Japan	Tokyo Traders
NULL	NULL	NULL	Japan	Mayumi's
NULL	NULL	NULL	USA	New Orleans Cajun Delights
NULL	NULL	NULL	USA	Grandma Kelly's Homestead
Elizabeth	Lincoln	Canada	NULL	NULL
Laurence	Lebihan	France	NULL	NULL
Frédérique	Citeaux	France	NULL	NULL
Hanna	Moos	Germany	NULL	NULL
Maria	Anders	Germany	NULL	NULL
Ana	Trujillo	Mexico	NULL	NULL
Antonio	Moreno	Mexico	NULL	NULL
Martin	Sommer	Spain	Spain	Cooperativa de Quesos 'Las Cabras'
Christina	Berglund	Sweden	Sweden	PB Knäckebröd AB
Thomas	Hardy	UK	UK	Exotic Liquids
Thomas	Hardy	UK	UK	Specialty Biscuits, Ltd.

Operation Sheet-1.6 Working with union operator

Operation title: Working with union operator

Purpose: To show functionalities of Join, left join, right join and full join

Equipment tools and materials: SQL server 2012

Step 1: Use the above merchant database from operation sheet 1.1

Task 1

Step 1: To list all unique countries for customers and suppliers we can write the following statement

- SELECT Country
  FROM Customers
  UNION
  SELECT Country
  FROM Supplier

Quality Criteria: your output should look like this

Country
Australia
Brazil
Canada
France
Germany
Japan
Mexico
Spain
Sweden
UK
USA

Operation Sheet-1.7 Data Control Language

Operation title: Data Control Language

Purpose: To show grant and revoke commands for a database user

Equipment tools and materials: SQL server 2012

Step 1: Use the above merchant database from operation sheet 1.1

Step 2: Imagine we have two database administrators ababe and kebede and we want them to create a table, insert and delete a data from the tables from the merchant database. So, we can write the following query (suppose that kebede will grant permission for ababe)

- CREATE login ababe with password='123'
  CREATE user ababe for login ababe
  GRANT create table to ababe
- CREATE login kebe with password='123'
  CREATE user kebe for login kebe
  GRANT insert,delete on database:: merchant to ababe

Step 3: Cancel ababe's permission on the merchant database

- REVOKE insert,delete on database:: merchant to ababe

Lap Test

Instruction: create a database and do the following tasks accordingly

Task 1: Create a database called ABC and create the following tables with their respective relation.

Customer Table

CUST_CODE	CUST_NAME	CUST_CITY	WORKING_AREA	CUST_COUNTRY	GRADE	PHONE_NO	AGENT_CODE
C00001	Micheal	New York	New York	USA	2	CCCCCCC	A008
C00002	Bolt	New York	New York	USA	3	DDNRDRH	A008
C00013	Holmes	London	London	UK	2	BBBBBBB	A003
C00015	Stuart	London	London	UK	1	GFSGERS	A003
C00018	Fleming	Brisban	Brisban	Australia	2	NHBGVFC	A012
C00019	Yearannaidu	Chennai	Chennai	India	1	ZZZZBFV	A010
C00020	Albert	New York	New York	USA	3	BBBBSBB	A008
C00021	Jacks	Brisban	Brisban	Australia	1	WERTGDF	A003
C00024	Cook	London	London	UK	2	FSDDSDF	A011
C00025	Ravindran	Bangalore	Bangalore	India	2	AVAVAVA	A011

Orders Table

ORD_NUM	ORD_AMOUNT	ADVANCE_AMOUNT	ORD_DATE	CUST_CODE	AGENT_CODE	ORD_DESCRIPTION
200100	1000.00	600.00	2008-08-01	C00013	A003	SOD
200102	2000.00	300.00	2008-05-25	C00013	A012	SOD
200107	4500.00	900.00	2008-08-30	C00015	A010	SOD
200110	3000.00	500.00	2008-04-15	C00019	A010	SOD
200112	2000.00	400.00	2008-05-30	C00021	A007	SOD
200113	4000.00	600.00	2008-06-10	C00025	A003	SOD
200114	3500.00	2000.00	2008-08-15	C00001	A008	SOD

Agent Table

AGENT_CODE	AGENT_NAME	WORKING_AREA	COMMISSION	PHONE_NO	COUNTRY
A003	Alex	London	0.13	075-12458969	
A007	Ramasundar	Bangalore	0.15	077-25814763	
A008	Alford	New York	0.12	044-25874365	
A010	Santakumar	Chennai	0.14	007-22388644	
A011	Ravi Kumar	Bangalore	0.15	077-45625874	
A012	Lucida	San Jose	0.12	044-52981425	

Task 2: Retrieve the names of customers who have placed orders with an order amount greater than the average order amount.

Task 3: Retrieve the agent names in alphabetical order.

Task 4: Retrieve the orders where the order amount is either greater than $5000 or the advance amount is less than $1000.

Task 5: Retrieve the distinct agent codes from the AGENTS table and the distinct agent codes from the ORDERS table.

Task 6: Retrieve the customer names and their corresponding working areas.

Unit Two: SQL statements with functions

This unit is developed to provide you the necessary information regarding the following content coverage and topics

- Arithmetic operations
- String functions and operators
- Mathematical functions
- Date functions

This unit will also assist you to attain the learning outcomes stated in the cover page. Specifically, upon completion of this learning guide, you will be able to:

- Understand and use arithmetic operations
- Use string functions and operators
- Understand and use mathematical functions
- Understand and use date functions

2.1. Arithmetic operations

Arithmetic operators can perform arithmetical operations on numeric operands involved. Arithmetic operators are addition (+), subtraction (-), multiplication (*) and division (/).

Operator	Meaning
+(Add)	Addition
-(Subtract)	Subtraction
*(Multiply)	Multiplication
/(Divide)	Division
%(Modulo)	Returns the integer reminder of a division

Arithmetic operations syntax:

- SELECT column_name arithmetic operator FROM [table_name] WHERE [expression];

2.2. String functions and operators

The following table shows some string functions. All string functions works with the select statement.

Function	Description
ASCII	Returns the ASCII for the specified character
CHAR	Returns the character based on ASCII
CHARINDEX	Returns the position of a substring in a string
DATALENGTH	Returns the number of bytes used to represent an expression
DIFFERENCE	Compare two SOUNDEX values and returns integer value
LEFT	Extract a number of characters by starting from the left
UPPER	Convert a string in to upper case
LTRIM	Remove leading spaces from a string
SUBSTRING	Extract some characters from a string
REPLICATE	Replicates a string a specified number of times

- **The ASCII ()** function returns the ASCII value for the specific character.

ASCII syntax

- ASCII (character)

Character parameter returns the ASCII value. If more than one character is entered, it will only return the value for the first character

- **The CHAR ()** function returns the character based on the ASCII code.

CHAR syntax

- CHAR (code)

Code parameter returns the ASCII number code for the character

- **The CHARINDEX ()** function searches for a substring in a string, and returns the position. If the substring is not found, this function returns 0.

Note: This function performs a case-insensitive search.

CHARINDEX syntax

- CHARINDEX (substring, string, start)

Substring parameter is the one to search for

String parameter is the one to be searched

Start parameter is optional where the search will start

- **The DATALENGTH ()** function returns the number of bytes used to represent an expression.

Note: The DATALENGTH () function counts both leading and trailing spaces when calculating the length of the expression.

DATALENGTH syntax

- DATALENGTH (expression)

Expression parameter is the one that return data type length

- **The DIFFERENCE ()** function compares two SOUNDEX values, and returns an integer. The integer value indicates the match for the two SOUNDEX values, from 0 to 4.

0 indicates weak or no similarity between the SOUNDEX values. 4 indicate strong similarity or identically SOUNDEX values.

DIFFERENCE syntax

- DIFFERENCE (expression, expression)

The two expressions are to be compared. Can be contrast, variable or column

- **The LEFT ()** function extracts a number of characters from a string (starting from left).

LEFT syntax

- LEFT (string, number_of_chars)

String operator is the one to extract from

Number_of_chars operator, it is the number of characters to extract If the number exceeds the number of characters in string, it returns string

- **The UPPER ()** function converts a string to upper-case.

UPPER syntax

- UPPER (text)

Text parameter is the string to be converted

- **The LTRIM ()** function removes leading spaces from a string.

Note: Also look at the RTRIM () function.

LTRIM syntax

- LTRIM (string)

String parameter is the one to remove leading spaces from.

- **The SUBSTRING ()** function extracts some characters from a string.

SUBSTRING syntax

- SUBSTRING (string, start, length)

String parameter is the one the string to be extracted from

Start parameter is the first position in string is 1

Length parameter is number of characters to extract. Must be a positive number

- **The REPLICATE ()** function repeats a string a specified number of times.

REPLICATE syntax

- REPLICATE (string, integer)

String parameter is the string to repeated

Integer parameter is the number of times to repeat the string

2.3. Mathematical functions

The following table shows some mathematical functions. All string functions works with the SELECT statement.

Function	**Description**
COUNT	Returns the number of records returned by a select query
AVG	Returns the average value of an expression
DEGREES	Converts a value in radians to degrees
MAX	Returns the maximum value in a set of values
SQUARE	Returns the square of a number
SUM	Calculates the sum of a set of values
POWER	Calculates the sum of a set of values
TAN	Returns the tangent of a number
COS	Returns cosine of a number
CEILING	Returns the smallest integer value that is >= a number

- **The COUNT ()** function returns the number of records returned by a select query.

Note: NULL values are not counted.

COUNT () Syntax

- COUNT (expression)

Expression parameter is a field or a string value

- **The AVG ()** function returns the average value of an expression.

Note: NULL values are ignored.

AVG () syntax

- AVG (expression), the Expression parameter is a numeric value

- **The DEGREES ()** function converts a value in radians to degrees.

DEGREES () syntax

- DEGREES (number)

Number parameter is a numeric value

- **The MAX ()** function returns the maximum value in a set of values.

MAX () syntax

- MAX (expression)

Expression parameter is a numeric value

- **The SQUARE ()** function returns the square of a number.

SQUARE () syntax

- SQUARE (number)

Number parameter is a positive number to calculate the square

- **The SUM ()** function calculates the sum of a set of values.

Note: NULL values are ignored.

SUM () syntax

- SUM (expression)

Expression parameter is a field or a formula

- **The POWER ()** function returns the value of a number raised to the power of another number.

POWER () syntax

- POWER (a, b)

A parameter is a number (the base) and B parameter is a number (the exponent)

- **The TAN ()** function returns the tangent of a number.

TAN () syntax

- TAN (number)

Number parameter is a numeric value

- **The COS ()** function returns the cosine of a number.

COS () syntax

- COS (number)

Number parameter is a numeric value

- **The CEILING ()** function returns the smallest integer value that is larger than or equal to a number.

CEILING () syntax

- CEILING (number)

Number parameter is a numeric value

2.4. Date functions

The following table shows some date functions

Function	**Description**
CURRENT_TIMESTAMP	Returns the current date and time
DATEADD	Adds a time/date interval to a date and then returns the date
DATEDIFF	Returns the difference between two dates
DATEPART	Returns a single part of a date/time
CONVERT	Displays date/time data in different formats
GETDATE	Returns the current database system date and time

- The **CURRENT_TIMESTAMP** function returns the current date and time, in a 'YYYY-MM-DD hh:mm:ss.mmm' format.

CURRENT_TIMESTAMP syntax

  - CURRENT_TIMESTAMP

- **The DATEADD ( )** function adds or subtracts a time/date interval to a date and then returns the date.

DATEADD () syntax

  - DATEADD (datepart, number, date)

Interval parameter is to add time/date interval. Can be one of the following values:

  - ✓ yyyy, yy = Year
  - ✓ qq, q = Quarter
  - ✓ mm, m = month
  - ✓ dy, y = Day of the year
  - ✓ day, dd, d = Day
  - ✓ ww, wk = Week
  - ✓ dw, w = Weekday
  - ✓ hh = hour
  - ✓ mi, n = Minute
  - ✓ ss, s = Second
  - ✓ ms = Millisecond
  - Number parameter is the number of intervals to add to date. Can be positive (to get dates in the future) or negative (to get dates in the past)
  - Date parameter is the date that will be modified.

- **The DATEDIFF ()** function returns the difference between two dates.

DATEDIFF () syntax

  - DATEDIFF (date_part, start_date, end_date)

Interval parameter is the part to be returned

- ✓ yyyy, yy = Year
- ✓ qq, q = Quarter
- ✓ mm, m = month
- ✓ dy, y = Day of the year
- ✓ dd, d = Day
- ✓ ww, wk = Week
- ✓ dw, w = Weekday
- ✓ hh = hour
- ✓ mi, n = Minute
- ✓ ss, s = Second
- ✓ ms = Millisecond

➢ start date and end date parameters are the two dates to calculate the difference between.

- **The DATEPART ()** function is used to return a single part of a date/time, such as year, month, day, hour, minute, etc.

DATEPART () syntax

➢ DATEPART (datepart,date)

Where date is a valid date expression and datepart can be one of the following:

Interval parameter is the part to be returned

- ✓ yyyy, yy = Year
- ✓ qq, q = Quarter
- ✓ mm, m = month
- ✓ dy, y = Day of the year
- ✓ dd, d = Day
- ✓ ww, wk = Week
- ✓ dw, w = Weekday
- ✓ hh = hour
- ✓ mi, n = Minute
- ✓ ss, s = Second

- ✓ ms = Millisecond

- **The CONVERT ()** function is a general function that converts an expression of one data type to another.

The CONVERT () function can be used to display date/time data in different formats.

CONVERT () syntax

- CONVERT (data_type(length),expression,style)
  - ✓ data_type(length) Specifies the target data type (with an optional length)
  - ✓ expression Specifies the value to be converted
  - ✓ style Specifies the output format for the date/time

- **The GETDATE ()** function returns the current date and time from the SQL Server.

GETDATE () syntax

- GETDATE ()

Self-Check 2

Part I: Choose the best answer

_____ 1. Which of the following is an example of an arithmetic operator?

a) CONCATENATE c) SUBSTRING

b) LENGTH d) DIVIDE

_____ 2. Which function is used to convert a string to uppercase in SQL?

a) LOWER() c) TRIM()

b) UPPER() d) CONCAT()

_____ 3. Which function is used to return the current date and time in SQL?

a) YEAR() c) DAY()

b) CURRENT_TIMESTAMP d) NOW()

_____ 4. Which mathematical function in SQL returns the smallest integer value that is >= a number

a) ROUND() c) CEILING()

b) FLOOR () d) ABS()

Part II: Say true or false

_____ 1. The CONCAT() function in SQL is used to concatenate two or more strings.

_____ 2. The DATEDIFF() function in SQL is used to calculate the difference between two dates.

_____ 3. LTRIM function in SQL is used to create leading spaces from a string

_____ 4. REPLICATE() function repeats a string a specified number of times.

Operation Sheet-2.1 Arithmetic operators and string function

Operation title: Arithmetic operators and string function

Purpose: To show functionalities of arithmetic operators and string functions in a table

Equipment tools and materials: SQL server 2012

Step 1: Use merchant database from operation sheet 1.1

Step 2: Calculate the total amount (UnitPrice * Quantity) for each order item and display the result with the order item ID and the total amount.

- SELECT Id, UnitPrice * Quantity AS TotalAmount

FROM OrderItem;

Quality Criteria: your output should look like this

Id	TotalAmount
1	168.00
2	98.00
3	174.00
4	167.40
5	1696.00
6	77.00
7	1484.00
8	252.00
9	100.80
10	234.00

Step 3: To retrieve the product names that start with the letter "A" and display them in uppercase.

- SELECT UPPER(ProductName) AS ProductName
  FROM Product
  WHERE ProductName LIKE 'A%';

Quality Criteria: your output should look like this

ProductName
ANISEED SYRUP

Operation Sheet-2.2 Date function

Operation title: Date function

Purpose: To show functionalities of some date functions

Equipment tools and materials: SQL server 2012

Step 1: Use merchant database from operation sheet 1.1

Step 2: let us create a table called orders which stores orders of customers with their order id, product name and the date, which the order is placed. We can write the table as shown below

- CREATE TABLE Orders
  (
  OrderId int NOT NULL PRIMARY KEY IDENTITY,

ProductName varchar(50) NOT NULL,
OrderDate datetime DEFAULT GETDATE()

)

Step 3: to insert order data in the orders table we can write the following statement

- INSERT INTO Orders1111 (ProductName) VALUES ('Shola Milk')

Quality Criteria: your output should look like this

OrderId	ProductName	OrderDate
1	Shola Milk	2023-11-30 00:06:11.550

Step 4: From the previous table, if we want to retrieve the date in year, month, date format, we can write the following query

- SELECT DATEPART(yyyy,OrderDate) AS OrderYear,
  DATEPART(mm,OrderDate) AS OrderMonth,
  DATEPART(dd,OrderDate) AS OrderDay
  FROM Orders1111
  WHERE OrderId=1

Quality Criteria: your output should look like this

OrderYear	OrderMonth	OrderDay
2023	11	30

Step 5: If we want to add 30 days to the "OrderDate", to find the payment date. We can write the following query

- SELECT OrderId,DATEADD(day,30,OrderDate) AS OrderPayDate
  FROM Orders1111

Quality Criteria: your output should look like this

OrderId	OrderPayDate
1	2023-12-30 00:06:11.550

Lap Test

Instruction: Use ABC database (from unit one lap test) in order to perform the following tasks

Task 1: Retrieve the total number of products

Task 2: Calculate the total amount (ORD_AMOUNT + ADVANCE_AMOUNT) for each order and display the result with the order number and the total amount.

Task 3: Calculate the average commission for all agents and display the result rounded to two decimal places.

Unit Three: SQL statements with aggregation and filtering

This unit is developed to provide you the necessary information regarding the following content coverage and topics

- Function of group by statement
- Function of having clause
- Backup database

This unit will also assist you to attain the learning outcomes stated in the cover page.
Specifically, upon completion of this learning guide, you will be able to:

- Use group by statement to aggregate data
- Sort aggregated data
- Filter aggregated data using the 'having' clause
- Backup a database

3.1. Function of group by statement

The GROUP BY statement groups rows that have the same values into summary rows. It is often used with aggregate functions (COUNT (), MAX (), MIN (), SUM (), AVG ()) to group the result-set by one or more columns.

The GROUP BY Statement in SQL is used to arrange identical data into groups with the help of some functions. i.e., if a particular column has the same values in different rows, then it will arrange these rows in a group.

Features

- GROUP BY clause is used with the SELECT statement.
- In the query, the GROUP BY clause is placed after the WHERE clause.
- In the query, the GROUP BY clause is placed before the ORDER BY clause if used.
- In the query, the Group BY clause is placed before the Having clause.
- Place condition in the having clause.

GROUP BY Syntax

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
ORDER BY column_name(s);
```

Example: let's say we have a customers table with the following data

OrderID	CustomerID	EmployeeID	OrderDate	Country
10248	90	5	1996-07-04	Adama
10249	81	6	1996-07-05	Welkite
10250	34	4	1996-07-08	Adama

If we want the number of customers in each country, sorted high to low: we can write the following query

- SELECT COUNT (CustomerID), Country
  FROM Customers
  GROUP BY Country
  ORDER BY COUNT (CustomerID) DESC;

I. GROUP BY With JOIN

Below there is a customers table with the following data

OrderID	CustomerID	EmployeeID	OrderDate	ShipperID
10248	90	5	1996-07-04	3
10249	81	6	1996-07-05	1
10250	34	4	1996-07-08	2

And here there is a shippers table with the following data

ShipperID	ShipperName
1	Speedy Express
2	United Package
3	Federal Shipping

If we want to retrieve the number of orders sent by each shipper, we can write the following SQL statement

- SELECT Shippers.ShipperName, COUNT
  (Orders.OrderID) AS NumberOfOrders from Orders

LEFT JOIN Shippers ON Orders.ShipperID = Shippers.ShipperID

GROUP BY ShipperName;Sorting aggregated data

II. GROUP BY With ORDER BY CLAUSE

you can sort aggregate data using the ORDER BY clause in combination with the aggregate functions. Sorting aggregate data in SQL Server involves using aggregate functions to calculate summary values and then sorting the result set based on those calculations

3.2. Function of having clause

The HAVING clause was introduced in SQL to allow the filtering of query results based on aggregate functions and groupings, which cannot be achieved using the WHERE clause that is used to filter individual rows.

In simpler terms MYSQL, the HAVING clause is used to apply a filter on the result of GROUP BY based on the specified condition. The conditions are Boolean type i.e. use of logical operators (AND, OR). This clause was included in SQL as the WHERE keyword failed when we use it with aggregate expressions. Having is a very generally used clause in SQL. Similar to WHERE it helps to apply conditions, but HAVING works with groups. If you wish to filter a group, the HAVING clause comes into action.

Some important points:

- Having clause is used to filter data according to the conditions provided.
- Having a clause is generally used in reports of large data.
- Having clause is only used with the SELECT clause.
- The expression in the syntax can only have constants.
- In the query, ORDER BY is to be placed after the HAVING clause, if any.
- HAVING Clause is implemented in column operation.
- Having clause is generally used after GROUPBY.

The main difference between where and having clause

Where Clause in SQL	Having Clause in SQL
Applied before GROUP BY clause	Used after GROUP BY clause
Used with single row operations such as Upper, Lower and so on	Applicable with multiple row functions such as Sum, count and so on

HAVING clause syntax

SELECT col_1, function_name(col_2)

FROM tablename

WHERE condition

GROUP BY column1, column2

HAVING Condition

ORDER BY column1, column2

Example: Below there is a customer table

Customer ID	CustomerName	ContactName	Address	City	Postal Code	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK

5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

If we want to lists the number of customers in each country. And only include countries with more than 5 customers: the SQL statement will be

- SELECT COUNT (CustomerID), Country
  FROM Customers
  GROUP BY Country
  HAVING COUNT (CustomerID) > 5;

If we want to lists the number of customers in each country, sorted high to low (Only include countries with more than 5 customers): the SQL statement will be

- SELECT COUNT (CustomerID), Country
  FROM Customers
  GROUP BY Country
  HAVING COUNT (CustomerID) > 5
  ORDER BY COUNT (CustomerID) DESC;

3.4. Backup database

The BACKUP DATABASE statement is used in SQL Server to create a full back up of an existing SQL database.

Syntax

BACKUP DATABASE databasename
TO DISK = 'filepath';

- **Backup with differential**

A differential backup reduces the backup time (since only the changes are backed up).

The following SQL statement creates a differential back up of the database "testDB":

Example: BACKUP DATABASE testDB TO DISK = 'D:\backups\testDB.bak'
WITH DIFFERENTIAL;

Self-Check 3

Part I: Choose the best answer

_____ 1. What is the primary function of the GROUP BY statement in SQL?

a) It sorts the data in ascending order.

b) It filters data based on specified conditions.

c) It groups data based on one or more columns.

d) It performs mathematical calculations on the data.

_____ 2. You can sort aggregated data in a SQL query result using?

a) GROUP BY clause c) WHERE clause

b) ORDER BY clause d) HAVING clause

_____ 3. What is the main purpose of the HAVING clause in SQL?

a) To filter rows before they are grouped.

b) To sort groups in descending order.

c) To filter groups after they have been created.

d) To perform calculations on individual records.

_____ 4. When backing up a database, what does it involve?

a) Creating a duplicate copy of the database.

b) Deleting unnecessary records from the database.

c) Rearranging the data within the database.

d) Upgrading the database to a newer version.

Part II: Say true or false

_____ 1. The GROUP BY statement is used to group rows based on specified columns

_____ 2. SORT BY is used to arrange individual records in a database.

_____ 3. The HAVING clause is applied before the GROUP BY clause in a SQL query.

_____ 4. Backup files should be stored in a separate location from the original database.

Operation Sheet-3.1 Sort aggregated data and backup

Operation title: Sort aggregated data and backup database

Purpose: To sort aggregated data using group by, order by and having clause. And to be able to back up the database.

Equipment tools and materials: SQL server 2012

Step 1: Use merchant database from operation sheet 1.1

Step 2: To retrieve the total order amount for each customer and sort the result in descending order of the sum.

```
SELECT CUST_CODE, SUM(ORD_AMOUNT) AS TotalOrderAmount
FROM ORDERS
GROUP BY CUST_CODE
ORDER BY TotalOrderAmount DESC;
```

Quality Criteria: your output should look like this

CUST_CODE	TotalOrderAmount
C00015	4500.00
C00025	4500.00
C00020	4000.00
C00001	3500.00
C00019	3000.00
C00013	3000.00
C00002	2500.00
C00021	2000.00

Step 3: To list the number of products for each supplier, sorted high to low.

```
SELECT S.CompanyName, COUNT(P.Id) AS Products
FROM Supplier S
JOIN Product P ON S.Id = P.SupplierId
GROUP BY S.CompanyName
ORDER BY COUNT(P.Id) DESC
```

Quality Criteria: your output should look like this

CompanyName	Products
Exotic Liquids	3
Grandma Kelly's Homestead	3
New Orleans Cajun Delights	2
Tokyo Traders	2

Step 4: List all countries with more than 1 suppliers.

```
SELECT Country, COUNT(Id) AS Suppliers
FROM Supplier
GROUP BY Country
HAVING COUNT(Id) > 1
```

Quality Criteria: your output should look like this

Country	Suppliers
Japan	2
UK	2
USA	2

Step 5: To perform full back up the merchant database we can write the following query (but first we have to create backups folder in the specified location)

```
BACKUP DATABASE merchant
TO DISK = 'D:\backups\testDB.bak';
```

Lap Test

Instruction: Use ABC database (from unit one lap test) in order to perform the following tasks

Task 1: Retrieve the total order amount for each customer and display the result with the customer code and their total order amount.

Task 2: Retrieve the agents who have a commission greater than 5,000 and display their agent code and commission.

Task 3: Create a full backup of the database ABC and specify the backup file path as "C:\Backup\ABC.bak".

Reference

Books

SQL Server Transaction Log Management by Tony Davis and Gail Shaw 1st edition

The 45 Database Performance Tips for Developers by Phil Factor, Grant Fritchey 3rd edition

The 119 SQL Code Smells by Tony Davis and Gail Shaw 3rd edition

URL

https://www.freebookcentre.net/Database/Sql-Server-Books-Download.html

https://www.w3schools.com/sql/default.asp

https://www.geeksforgeeks.org/sql-ddl-dql-dml-dcl-tcl-commands/

https://www.geeksforgeeks.org/sql-where-clause/

https://www.geeksforgeeks.org/sql-order-by/

https://www.geeksforgeeks.org/sql-group-by/

https://www.geeksforgeeks.org/sql-having-clause-with-examples/

https://www.sqlshack.com/sql-queries-in-sql-server-a-beginners-guide/

https://www.w3schools.in/mysql/ddl-dml-dcl#:~:text=TCL%20stands%20for%20Transaction%20Control,treated%20as%20a%20single%20unit.

Developer's Profile

No	Name	Qualification	Field of Study	Organization/ Institution	Mobile number	E-mail
1	Frew Atkilt	M-Tech	Network & Information Security	Bishoftu Polytechnic College	0911787374	frew.frikii@gmail.com
2	Gari Lencha	MSc	ICT Managment	Gimbi Polytechnic	0917819599	Garilencha12@gmail.com
3	Kalkidan Daniel	BSc	Computer Science	Entoto Polytechnic	0978336988	kalkidaniel08@gmail.com
4	Solomon Melese	M-Tech	Computer Engineering	M/G /M/Polytechnic College	0918578631	solomonmelese6@gmail.com
5	Tewodros Girma	MSc	Information system	Sheno Polytechnic College	0912068479	girmatewodiros @gmail.com
6	Yohannes Gebeyehu	BSc	Computer Science	Entoto Polytechnic College	0923221273	yohannesgebeyehu73@gmail.com