Web Development and Database Administration

Level-III

Based on November 2023, Curriculum Version II



Module Title: Basic structured query language

Module code: EIS WDDBA3 M07 1123

Nominal duration: 100 Hours

Prepared by: Ministry of Labor and Skill

November, 2023

Addis Ababa, Ethiopia

የሥራና ክህሎት ሚኒስቴር
Ministry of Labor and Skills

Table of Contents

Acknowledgment 1
Acronym 2
Introduction to the Module 3
Unit One:Fundamental concepts of relational database 4
1.1. Data management approaches 5
1.2. Relational Database Management System 10
1.3. Structured query language 14
Self Check 1 20
Operation sheet 1.1: Install SQL server 2012 22
Lap Test 42
Unit Two: Data definition language 43
2.1. Introduction to SQL data definition language commands 44
2.2. Database planning 44
2.3. Usage of relevant naming convention for all database elements 45
2.4. Database structure creation and manipulation 54
Self-Check 2 66
Operation sheet 2.1: Create Database structure 67
Lap Test 70
Unit Three: Data manipulation language 71
3.1. Overview of SQL data manipulation language commands 72
3.2. Data insertion 72
3.3. Modification of existing data 74
3.4. Data deletion 76
Self check 3.1 79
Operation sheet 3.1 Data Insertion 80
Lap Test 83
Unit Four: Data query language 84

4.1. Overview of SQL data query language 85
4.2. Selection of data from a single table 86
4.3. Retrieval of data selectively 88
4.4. Working with functions 103
4.5. Working with subqueries 113
Self-check 4 115
Operation sheet 4.1 Inserting data into a database 116
Lap Test 119
Reference 120
Developer's Profile 121

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

SQL ---------- Structured query language

RDBMS ---------- Relational database management system

DDL ---------- Data definition language

DML ---------- Data manipulation language

DCL ---------- Data control language

DQL ---------- Data query language

DBA----------Database administrator

Introduction to the Module

This module covers all the fundamental concepts of SQL language, such as creating database and tables, using constraints, adding records to a table updating and deleting records in a table, and so on. Once you're familiar with the basics, you'll move on to next level that explains the methods of retrieving records, searching records in the table based on pattern, etc. Finally, you'll explore some advanced concepts. In this module, we delve into four critical units fundamental concepts of relational database, DDL, DML and data query language to equip you with the skills and knowledge necessary to enable you to be competent.

Module covers the units:

- Fundamental concepts of relational database
- Data definition language
- Data manipulation language
- Data query language

Learning outcomes of the module

- Describe data management approaches and database
- Practice installing and uninstalling DBMS software
- Create and manage databases using different DBMSs
- Explain structured query language and differentiate SQL keywords from identifiers
- Apply SQL DDL and DML to create and manipulate database structures
- Explain SQL data control language and differentiate between SQL DDL, DML, DQL and DCL

Module Instruction

For effective use of this module trainees are expected to follow the following module instruction:

1. Read the information written in each unit
2. Accomplish the Self-checks at the end of each unit
3. Perform Operation Sheets which were provided at the end of units
4. Do the “LAP test” given at the end of each unit

Unit One:Fundamental concepts of relational database

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

- Data management approaches
- Relational Database management system
- Structured query 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:

- Describe Data management approaches
- Understand properties, characteristics and benefits of a database
- Define relational database management system
- Describe the different types of RDBMS
- Install a DBMS software
- Explain Structured query language

1.1. Data management approaches

The way in which computers manage data has come a long way over the last few decades. Today's users take for granted the many benefits found in a database system. However, it wasn't that long ago that computer relied on a much less elegant and costly approach to data management called the file-based system.

1.1.1 File based approach

One way to keep information on a computer is to store it in permanent files. A company system has a number of application programs; each of them is designed to manipulate data files. These application programs have been written at the request of the users in the organization. New applications are added to the system as the need arises. The system just described is called the file-based system. Consider a traditional banking system that uses the file-based system to manage the organization's data. There are different departments in the bank. Each has its own applications that manage and manipulate different data files. For banking systems, the programs may be used to debit or credit an account, find the balance of an account, add a new mortgage loan and generate monthly statements.

Disadvantages of the file-based approach

Using the file-based system to keep organizational information has a number of disadvantages. Listed below are five examples.

- Data redundancy

Often, within an organization, files and applications are created by different programmers from various departments over long periods of time. This can lead to data redundancy, a situation that occurs in a database when a field needs to be updated in more than one table. This practice can lead to data inconsistency, a situation where various copies of the same data are conflicting, wastes storage space and duplicates effort.

- Data isolation

Data isolation is a property that determines when and how changes made by one operation become visible to other concurrent users and systems. This issue occurs in a concurrency situation. This is a problem because: It is difficult for new applications to retrieve the appropriate data, which might be stored in various files.

- Integrity problems

Problems with data integrity is another disadvantage of using a file-based system. It refers to the maintenance and assurance that the data in a database are correct and consistent. Factors to consider when addressing this issue are:

Data values must satisfy certain consistency constraints that are specified in the application programs.
It is difficult to make changes to the application programs in order to enforce new constraints.

- Security problems

Security can be a problem with a file-based approach because:
There are constraints regarding accessing privileges.
Application requirements are added to the system in an ad-hoc manner so it is difficult to enforce constraints.

- Concurrency access

Concurrency is the ability of the database to allow multiple users access to the same record without adversely affecting transaction processing. A file-based system must manage or prevent, concurrency by the application programs.
Typically, in a file-based system, when an application opens a file, that file is locked. This means that no one else has access to the file at the same time.
In database systems, concurrency is managed thus allowing multiple users access to the same record. This is an important difference between database and file-based systems.

1.1.2 Database Approach

The difficulties that arise from using the file-based system have prompted the development of a new approach in managing large amounts of organizational information called the database approach.
Most companies keep track of customer information by storing it in a database. This data may include customers, employees, products, orders or anything else that assists the business with its operations.

What is a database?

A database is a shared collection of related data used to support the activities of a particular organization. A database can be viewed as a repository of data that is defined once and then accessed by various users. With the database approach, we can have the traditional banking system where Personnel Department, the Account Department and the Loan Department access the shared corporate database.

1.1.3 Database Properties

A database has the following properties:

It is a representation of some aspect of the real world or a collection of data elements (facts) representing real world information.

A database is logical, coherent and internally consistent.

A database is designed, built and populated with data for a specific purpose.

Each data item is stored in a field. A combination of fields makes up a table. For example, each field in an employee table contains data about an individual employee. A database can contain many tables.

1.1.4 Characteristics and Benefits of a Database

There are a number of characteristics that distinguish the database approach from the file-based system or approach.

- Self-describing nature of a database system

A database system is referred to as self-describing because it not only contains the database itself, but also metadata which defines and describes the data and relationships between tables in the database. This information is used by the DBMS software or database users if needed. This separation of data and information about the data makes a database system totally different from the traditional file-based system in which the data definition is part of the application programs. Because the database contains a description of its own structure, it's self-describing. The database is integrated because it includes not only data items but also the relationships among data items. The database stores metadata in an area called the data dictionary, which describes the tables, columns, indexes, constraints, and other items that make up the database.

- Insulation between program and data

In the file-based system, the structure of the data files is defined in the application programs so if a user wants to change the structure of a file, all the programs that access that file might need to be changed as well.

On the other hand, in the database approach, the data structure is stored in the system catalogue and not in the programs. Therefore, one change is all that is needed to change the structure of a file. This insulation between the programs and data is also called program-data independence.

- Support for multiple views of data

A database supports multiple views of data. A view is a subset of the database, which is defined and dedicated for particular users of the system. Multiple users in the system might have different views of the system. Each view might contain only the data of interest to a user or group of users.

- Sharing of data and multiuser system

Current database systems are designed for multiple users. That is, they allow many users to access the same database at the same time. This access is achieved through features called concurrency control strategies. These strategies ensure that the data accessed are always correct and that data integrity is maintained. The design of modern multiuser database systems is a great improvement from those in the past which restricted usage to one person at a time.

- Control of data redundancy

In the database approach, ideally, each data item is stored in only one place in the database. In some cases, data redundancy still exists to improve system performance, but such redundancy is controlled by application programming and kept to minimum by introducing as little redundancy as possible when designing the database.

- Data sharing

The integration of all the data, for an organization, within a database system has many advantages. First, it allows for data sharing among employees and others who have access to the system. Second, it gives users the ability to generate more information from a given amount of data than would be possible without the integration.

- Enforcement of integrity constraints

Database management systems must provide the ability to define and enforce certain constraints to ensure that users enter valid information and maintain data integrity. A database constraint is a restriction or rule that dictates what can be entered or edited in a table such as a postal code using a certain format or adding a valid city in the City field.

There are many types of database constraints. Data type, for example, determines the sort of data permitted in a field, for example numbers only. Data uniqueness such as the primary key ensures that no duplicates are entered. Constraints can be simple (field based) or complex (programming).

- Restriction of unauthorized access

Not all users of a database system will have the same accessing privileges. For example, one user might have read-only access (i.e., the ability to read a file but not make changes), while another might have read and write privileges, which is the ability to both read and modify a file. For this reason, a database management system should provide a security subsystem to create and control different types of user accounts and restrict unauthorized access.

- Data independence

Another advantage of a database management system is how it allows for data independence. In other words, the system data descriptions or data describing data (metadata) are separated from the application programs. This is possible because changes to the data structure are handled by the database management system and are not embedded in the program itself.

- Transaction processing

A database management system must include concurrency control subsystems. This feature ensures that data remains consistent and valid during transaction processing even if several users update the same information.

- Backup and recovery facilities

Backup and recovery are methods that allow you to protect your data from loss. The database system provides a separate process, from that of a network backup, for backing up and recovering data. If a hard drive fails and the database stored on the hard drive is not accessible, the only way to recover the database is from a backup.

If a computer system fails in the middle of a complex update process, the recovery subsystem is responsible for making sure that the database is restored to its original state. These are two more benefits of a database management system.

1.2. Relational Database Management System

Relational Database Management System (RDBMS) is a type of Database Management System (DBMS) based on the relational model developed by E. F. Codd. It serves as the foundation for SQL and various modern database systems such as MS SQL Server, IBM DB2, Oracle, MySQL, PostgreSQL, and Sybase.

Purpose of DBMS: DBMS emerged in the late 1960s to assist in storing and managing data. Initially designed for mainframes, DBMS popularity extended to various computing platforms, including minicomputers, personal computers, workstations, and specialized servers.

Definition of DBMS:

A DBMS is a collection of programs enabling users to create, maintain, and control access to databases. Its primary goal is to provide a convenient and efficient environment for users to retrieve and store information.

Abstraction Layer:

The DBMS abstracts the physical details of database storage, allowing applications to focus on logical data characteristics. It organizes and structures data, defining its personality through a data model.

Data Model and SQL:

The data model, determined by a DBMS, dictates its characteristics and suitability for specific applications. SQL, a language for relational databases, aligns with the relational data model where data is organized as tables.

Schemas, Domains, and Constraints:

A database's structure is its schema, describing its complete logical view. Domains represent the possible values an attribute can assume, and constraints are rules governing attribute values.

Independence of Tables:

Relational databases offer flexibility as data resides in independent tables. Changes in one table do not affect others, provided there are no parent-child relationships.

Relations and Keys:

A relational database comprises one or more relations, essentially tables. Keys, such as primary keys, uniquely identify records. Foreign keys establish relationships between tables.

Primary Keys:

Primary keys uniquely identify each row and cannot have duplicate or NULL values. A table typically has one primary key, which may be a composite key if multiple fields are involved.

Foreign Keys:

Foreign keys link tables, referencing primary keys from another table. They establish relationships between tables, contributing to data integrity and consistency.

This overview provides a foundational understanding of RDBMS concepts, emphasizing their role in structuring, organizing, and managing data within a database system.

1.2.1 Requirement for Installing MS SQL Server DBMS Software

The following system requirements cover SQL Server 2012 Standard Edition on 32-bit and x64 platforms, as well as Itanium-based systems.

SQL Server 2012 64-bit Hardware Requirements:

Component	Minimum Requirement	Recommended Requirement
Processor	x64 Processor with at least 1.4 GHz	2.0 GHz or faster, 4 or more cores
RAM (64-bit)	1 GB for Express, 2 GB for all others	4 GB or more
Hard Disk Space (Database Engine)	3.5 GB minimum, 10 GB or more recommended	40 GB or more
Hard Disk Space (Analysis Services)	250 MB minimum, 2 GB or more recommended	40 GB or more
Hard Disk Space (Reporting Services)	3 GB minimum, 10 GB or more recommended	40 GB or more
Hard Disk Space (Integration Services)	1.5 GB minimum, 2.5 GB or more recommended	40 GB or more
Hard Disk Space (Full Installation)	3.8 GB minimum, 10 GB or more recommended	40 GB or more

SQL Server 2012 64-bit Software Requirements:

- **Operating System**: SQL Server 2012 supports a variety of Windows Server and Windows client operating systems. The specific editions of SQL Server 2012 (Enterprise, Standard, etc.) may have different requirements. It's important to check the official documentation for the exact operating system requirements for your specific edition.
- **.NET Framework**: SQL Server 2012 requires the .NET Framework. The installer will typically install the required version of .NET Framework for you.
- **SQL Server 2012 Edition**: Make sure you have the correct edition of SQL Server 2012 (e.g., Enterprise, Standard, and Express) for your needs and licensing.
- **SQL Server 2012 Service Pack**: It's advisable to install the latest service pack or cumulative update for SQL Server 2012 for bug fixes and enhancements.

Please keep in mind that these requirements are subject to change based on updates and service packs. Always refer to the official Microsoft SQL Server documentation for the most accurate and up-to-date information, especially if you have specific needs or configurations.

SQL Server 2012 32-bit Hardware Requirements:

Component	Minimum Requirement	Recommended Requirement
Processor	x86 or x64 Processor with at least 1.4 GHz	2.0 GHz or faster, 4 or more cores
RAM (32-bit)	1 GB for Express, 2 GB for all others	4 GB or more
Hard Disk Space (Database Engine)	2.2 GB minimum, 10 GB or more recommended	40 GB or more
Hard Disk Space (Analysis Services)	250 MB minimum, 2 GB or more recommended	40 GB or more
Hard Disk Space (Reporting Services)	2.1 GB minimum, 10 GB or more recommended	40 GB or more
Hard Disk Space (Integration Services)	1.5 GB minimum, 2.5 GB or more recommended	40 GB or more
Hard Disk Space (Full Installation)	2.5 GB minimum, 10 GB or more recommended	40 GB or more

SQL Server 2012 32-bit Software Requirements:

- **Operating System**: SQL Server 2012 32-bit supports a variety of 32-bit and 64-bit Windows Server and Windows client operating systems. The specific editions of SQL Server 2012 (Enterprise, Standard, etc.) may have different supported operating systems. Check the official documentation for exact details.
- **.NET Framework**: SQL Server 2012 requires .NET Framework 3.5 SP1 for installation. The installer will typically install this for you. You may also need .NET Framework 4.0 or

later for some features.

- **SQL Server 2012 Edition**: Make sure you have the correct edition of SQL Server 2012 (e.g., Enterprise, Standard, Express) for your needs and licensing.
- **SQL Server 2012 Service Pack**: It's advisable to install the latest service pack or cumulative update for SQL Server 2012 for bug fixes and enhancements.

1.3. Structured query language

SQL is a flexible language that you can use in a variety of ways. It's the most widely used tool for communicating with a relational database. SQL is a standard database language specifically designed for storing, retrieving, managing or manipulating the data inside a relational database management system (RDBMS). SQL became an ISO standard in 1987.

SQL is the most widely-implemented database language and supported by the popular relational database systems, like MySQL, SQL Server, and Oracle. However, some features of the SQL standard are implemented differently in different database systems. SQL is both a powerful language and one that is relatively easy to learn. SQL is both a de facto and an official standard language for database. management.

History of SQL

SQL was originally developed at IBM in the early 1970s. Initially it was called SEQUEL (Structured English Query Language) which was later changed to SQL (pronounced as S-Q-L). During the 1970s, a group at IBM San Jose Research Laboratory developed the System R relational database management system, based on the model introduced by Edgar F. Codd in his influential paper, A Relational Model of Data for Large Shared Data Banks.

SQL became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987. Since then, the standard has been revised to include a larger set of features. Despite the existence of such standards, though, most SQL code is not completely portable among different database systems without adjustments.

What You Can Do with SQL

There are lots more things you can do with SQL:

- You can create a database.
- You can create tables in a database.
- You can query or request information from a database.
- You can insert records in a database.
- You can update or modify records in a database.
- You can delete records from the database.
- You can set permissions or access control within the database for data security.
- You can create views to avoid typing frequently used complex queries.

The list does not end here; you can perform many other database-related tasks with SQL. You will learn about most of them in detail in upcoming chapters.

1.3.1 SQL key words

The SQL keywords are words that are reserved words that are not used as a user defined data. The most commonly used SQL key words according to ANSI/ISO SQL keywords are as follows:

ADA ALL AND ANY AS ASC AUTHORIZATION AVG BEGIN	DEC DECIMAL DECLARE DEFUALT DELETE DESC DISTNICT DOUBLE END	GRANT GROUP HAVING IN INDICATOR INSERT INT INTEGER INTO	NUMERIC OF ON OPEN OPTION OPEN OPTION OR ORDER
BETWEEN	ESCAPE	IS	PRIMARY
BY	EXEC	KEY	REAL
C	EXISTS	LANGAUGE	SELECT
CHAR	FETCH	LIKE	SET
CHARACTER	FLOAT	MAX	SOME
CHECK	FOR	MIN	SUM
CLOSE	FOREIGN	MODULE	TABLE
COBOL	FORTRAN	NOT	TO UNION
COMMIT	FOUND	NULL	UPDATE
CONTINUE	FROM		USER
COUNT	GO		VIEW
CREATE	GOTO		WHERE
CURRENT			WITH

CURSOR			WORK

1.3.2 Types of SQL Statements

Although SQL is considered a sublanguage because of its nonprocedural nature, it is nonetheless a complete language in that it allows you to create and maintain database objects, secure those objects, and manipulate the data within the objects. One common method used to categorize SQL statements is to divide them according to the functions they perform. Based on this method, SQL can be separated into three types of statements:

- **Data Definition Language (**DDL) statements are used to create, modify, or delete database objects such as tables, views, schemas, domains, triggers, stored procedures, and the database itself. The SQL keywords most often associated with DDL statements are CREATE, ALTER, and DROP. For example, you would use the CREATE TABLE statement to create a table, the ALTER TABLE statement to modify the table's properties, and the DROP TABLE statement to delete the table definition from the database.
- **Data Control Language** (DCL) protects your database from becoming corrupted. Used correctly, the DCL provides security for your database; the amount of protection depends on the implementation. If your implementation doesn't provide sufficient protection, you must add that protection to your application program. DCL statements allow you to control who or what (a database user can be a person or an application program) has access to specific objects in your database. With DCL, you can grant or restrict access by using the GRANT, Deny or REVOKE statements, the three primary DCL commands. The DCL statements also allow you to control the type of access each user has to database objects. For example, you can determine which users can view a specific set of data and which users can manipulate that data. DDL and DCL statements are commonly used by a database designer and database administrator for establishing the database structures used by an application.
- **Data Manipulation Language** (DML) allows a user or an application program to update the database by adding new data, removing old data, and modifying previously stored data. The primary keywords associated with DML statements are INSERT, UPDATE, and DELETE, all of which

represent the types of statements you'll probably be using the most. For example, you can use a update statement to modify existing data from a table and an insert statement to add data to a table.

- **Data query language (DQL)** When you need to retrieve data from a database, you use the SQL language to make the request. The DBMS processes the SQL request, retrieves the requested data, and returns it to you. This process of requesting data from a database and receiving back the results is called a database query—hence the name Structured Query Language. You use a SELECT statement to retrieve data from a table. The name Structured Query Language is actually somewhat of a misnomer. First of all, SQL is far more than a query tool, although that was its original purpose and retrieving data is still one of its most important functions. SQL is used to control all of the functions that a DBMS provides for its users, beyond what we have seen above.

1.3.3 What SQL Is and Isn't

The first thing to understand about SQL is that SQL isn't a procedural language, as are FORTRAN, BASIC, C, COBOL, Pascal, and Java. To solve a problem in one of those procedural languages, you write a procedure that performs one specific operation after another until the task is complete. The
procedure may be a linear sequence or may loop back on itself, but in either case, the programmer specifies the order of execution.

SQL, on the other hand, is nonprocedural. To solve a problem using SQL, simply tell SQL what you want instead of telling the system how to get what you want. The database management system (DBMS) decides the best way to get you what you request.

1.3.4 The Role of SQL

SQL is not itself a database management system, nor is it a stand-alone product. You cannot go into a computer store and "buy SQL." Instead, SQL is an integral part of a database management system, a language and a tool for communicating with the DBMS.

The database engine is the heart of the DBMS, responsible for actually structuring, storing, and retrieving the data in the database. It accepts SQL requests from other DBMS components, such as a forms facility, report writer, or interactive query facility, from user-written application programs, and even from other computer systems. SQL plays many different roles:

• SQL is an interactive query language. Users type SQL commands into an interactive SQL program to retrieve data and display it on the screen, providing a convenient, easy-to-use tool for ad hoc database queries.

• SQL is a database programming language. Programmers embed SQL commands into their application programs to access the data in a database. Both user-written programs and database utility programs (such as report writers and data entry tools) use this technique for database access.

• SQL is a database administration language. The database administrator responsible for managing a database uses SQL to define the database structure and control access to the stored data.

• SQL is a client/server language. Personal computer programs use SQL to communicate over a network with database servers that store shared data. This client/server architecture has become very popular for enterprise-class applications.

• SQL is an Internet data access language. Internet web servers that interact with corporate data and Internet applications servers all use SQL as a standard language for accessing corporate databases.

• SQL is a distributed database language. Distributed database management systems use SQL to help distribute data across many connected computer systems. The DBMS software on each system uses SQL to communicate with the other systems, sending requests for data access.

• SQL is a database gateway language. In a computer network with a mix of different DBMS products, SQL is often used in a gateway that allows one brand of DBMS to communicate with another brand.

SQL has thus emerged as a useful, powerful tool for linking people, computer programs, and computer systems to the data stored in a relational database.

Self Check 1

Part I: Choose the best answer

1. In a relational database, what is a primary key?
   A) A key that provides a unique identifier for each record in a table.
   B) key used to establish relationships between tables.
   C) A key that allows null values in the table.
   D) A key used for sorting records in ascending order.
2. What is the purpose of foreign keys in a relational database?
   A) To uniquely identify each record in a table.
   B) To establish relationships between tables.
   C) To enforce data type constraints.
   D) To automatically generate primary keys.
3. What is the primary purpose of a Relational Database Management System (RDBMS)?
   A) To perform complex mathematical calculations.
   B) To manage and organize files in a computer system.
   C) To store, retrieve, and manage structured data in tables.
   D) To create graphical user interfaces for applications.
4. In a relational database, what is a foreign key?
   A) A key that uniquely identifies each record in a table.
   B) A key used to establish relationships between tables.
   C) A key that is automatically generated by the database.
   D) A key used for sorting records in descending order.
5. What does the acronym SQL stand for in the context of relational databases?
   A) Simple Query Language
   B) Structured Query Language
   C) System Query Language
   D) Standardized Query Logic

6. Which normalization form ensures that there are no partial dependencies in a relational database?
    A) First Normal Form (1NF)
    B) Second Normal Form (2NF)
    C) Third Normal Form (3NF)
    D) Fourth Normal Form (4NF)

Part II: Say true or false

1. In a relational database, a foreign key is used to uniquely identify each record in a table.
2. Normalization is the process of organizing data to minimize redundancy and dependency.
3. Views in an RDBMS are physical tables that store data.
4. ACID properties (Atomicity, Consistency, Isolation, Durability) ensure the reliability of transactions in a relational database.
5. In a relational database, a primary key can contain NULL values.

Operation sheet 1.1: Install SQL server 2012

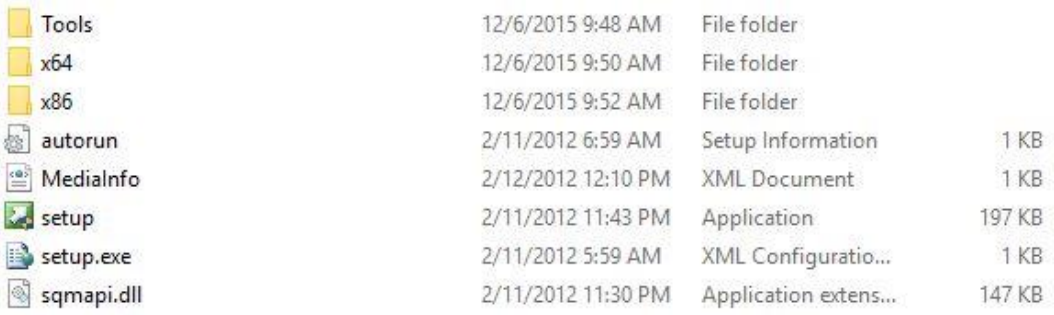
Operation title: SQL server 2012 installation

Purpose: To install SQL server 2012 installation

Equipment and Tools: SQL Server 2012

Steps to install SQL Server 2012

Step 1: Open installation media in new window and right click on setup file to run it “As Administrator”.



Step 2: Installation Center

After running the setup file, you’ll be redirected to Installation Media Center where you find various options. As we’re working on the installation, we won’t dig other parts. Click on Installation section and you’ll find something like the following window.

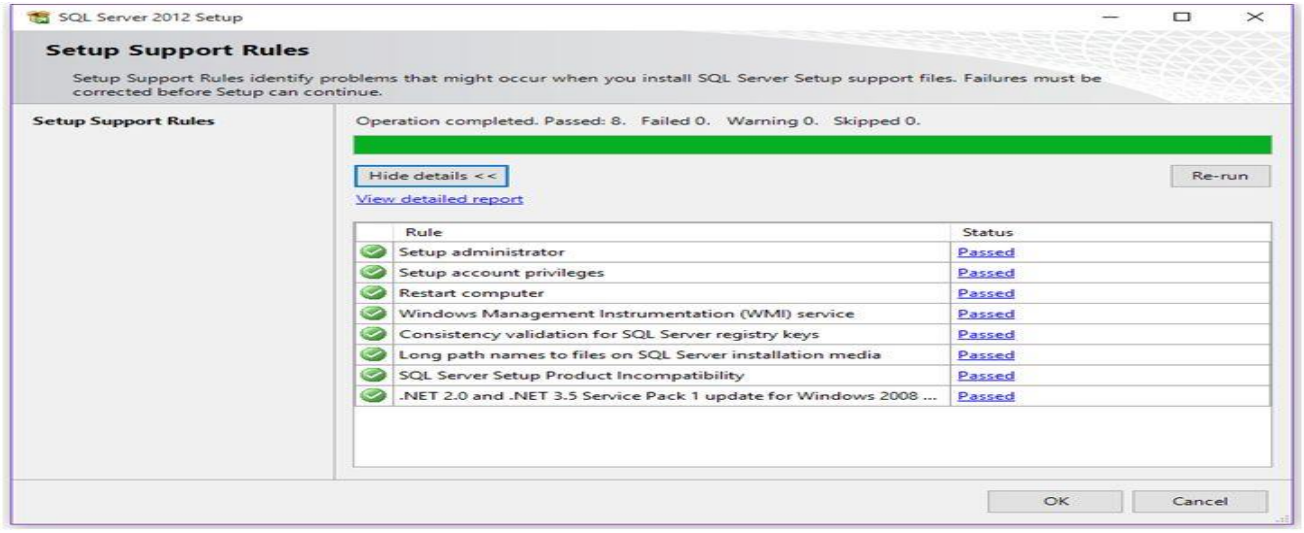


From here, you can perform stand-alone installation of SQL Server or you can add any additional features to installed instances of SQL Server. Also if you want to upgrade you version of SQL Server, there's an option for that as well.

As we want to perform stand-alone installation, we'll go with option 1. Click on the first link and the installation process begin.

Step 3: Setup Support Rules

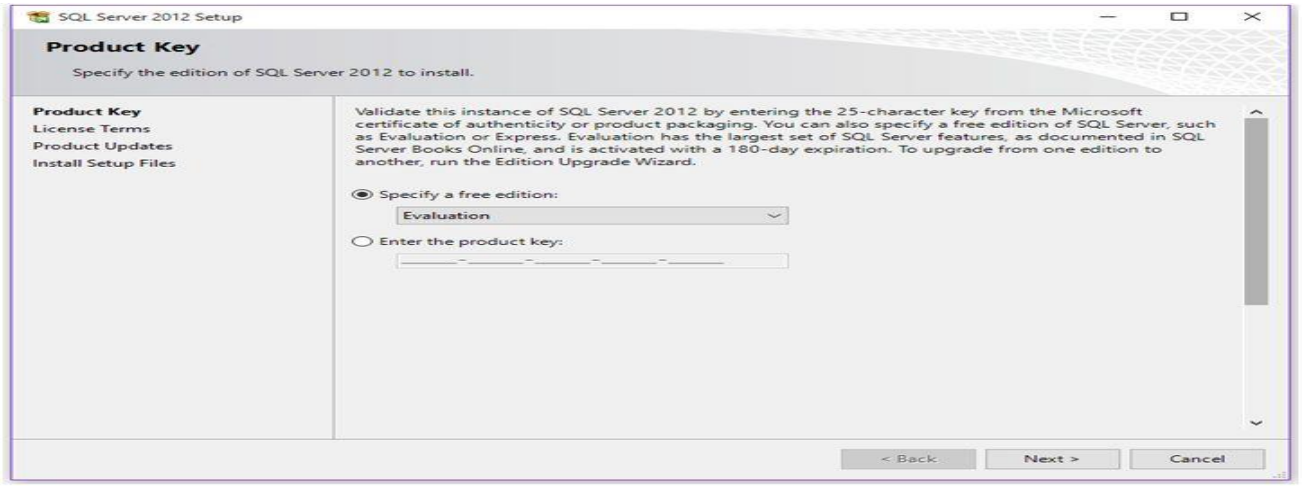
Before proceeding with the installation steps, SQL Server setup runs a setup to check all the things required for installation. This check is nothing but a kind of verification to ensure you can proceed further or not.



If any of the check fails, you'll get a failed notification in status column and you won't be able to proceed further with the installation. If all requirements fulfill, you'll get passed in Status column. Click OK.

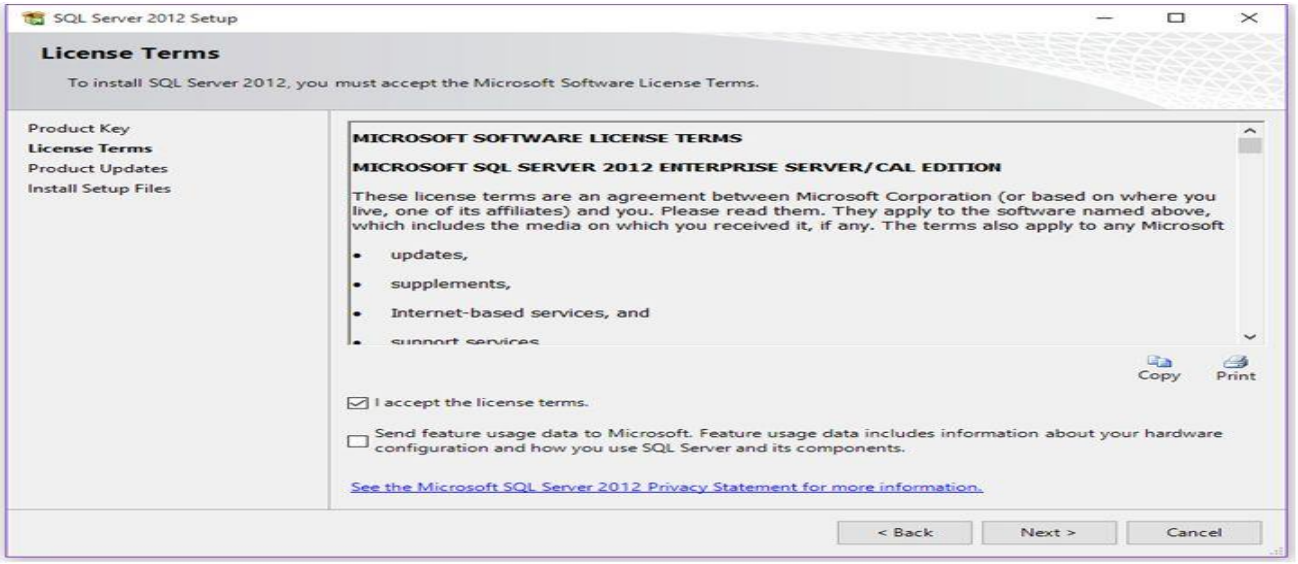
Step 4: Product Key

Select edition of SQL Server you want to install on your machine with your product key and click Next.



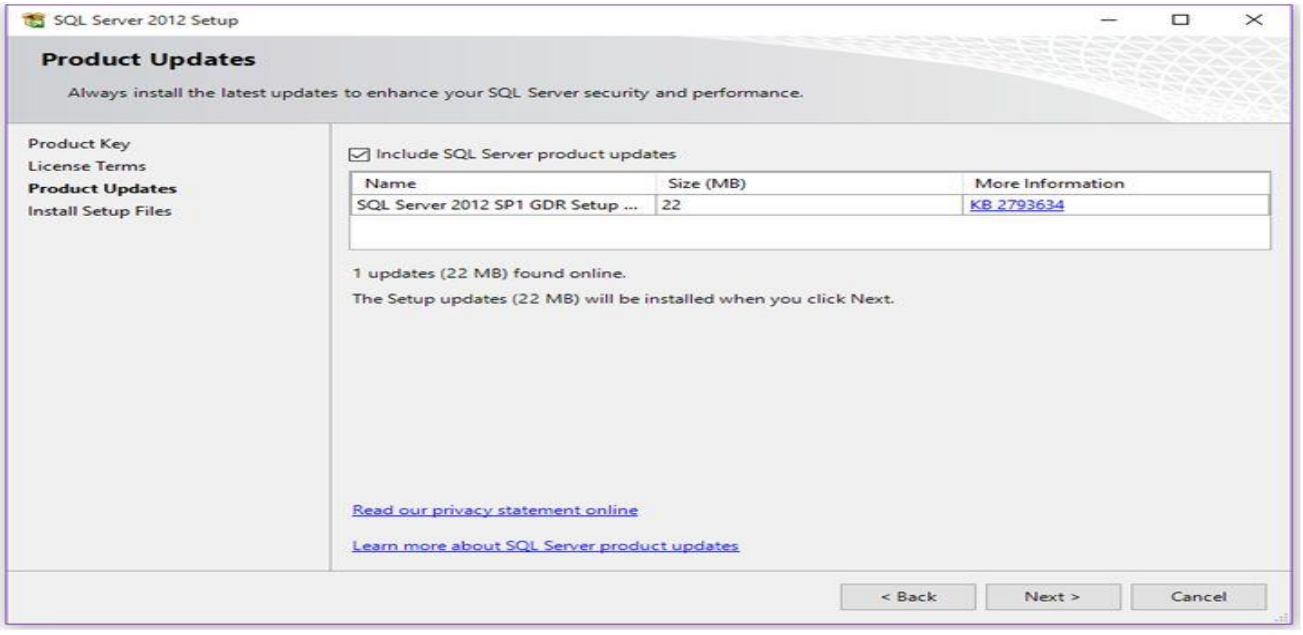
Step 5: License Term

Accept the license by clicking on “I accept license terms.” Click Next.



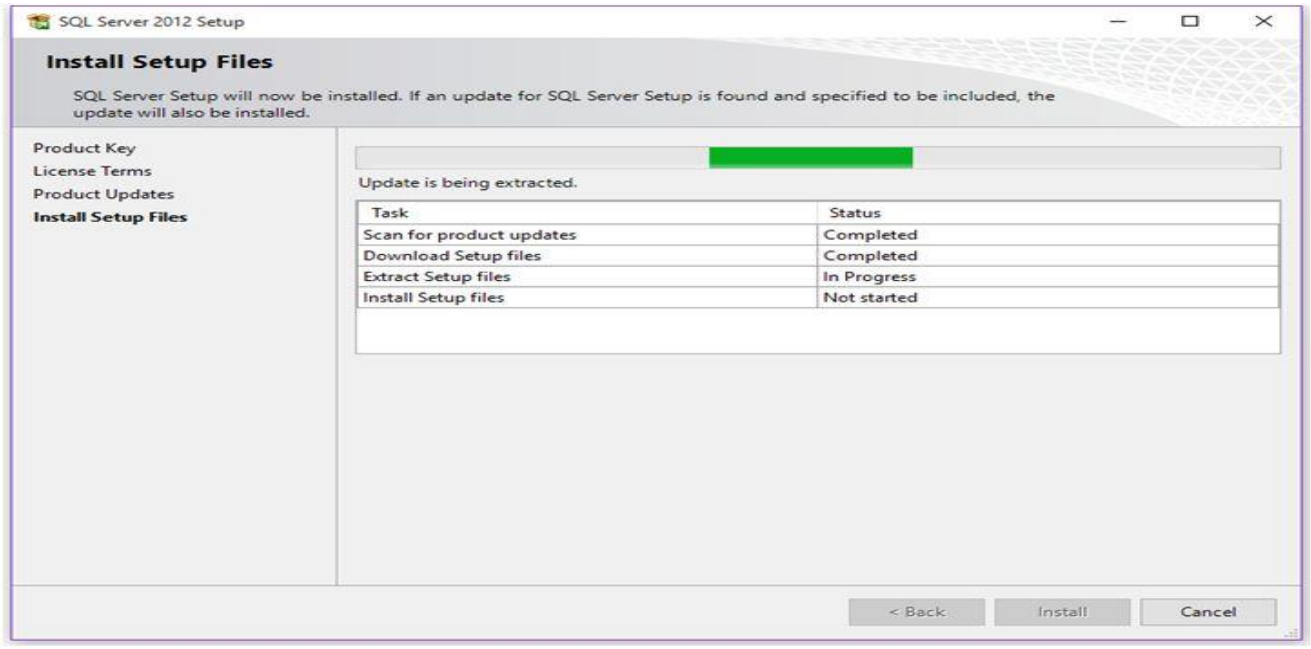
Step 6: Product Updates

Here, setup will look for latest product updates to enhance SQL Server performance as my setup found one update of 22 MB for enhancement as shown below.



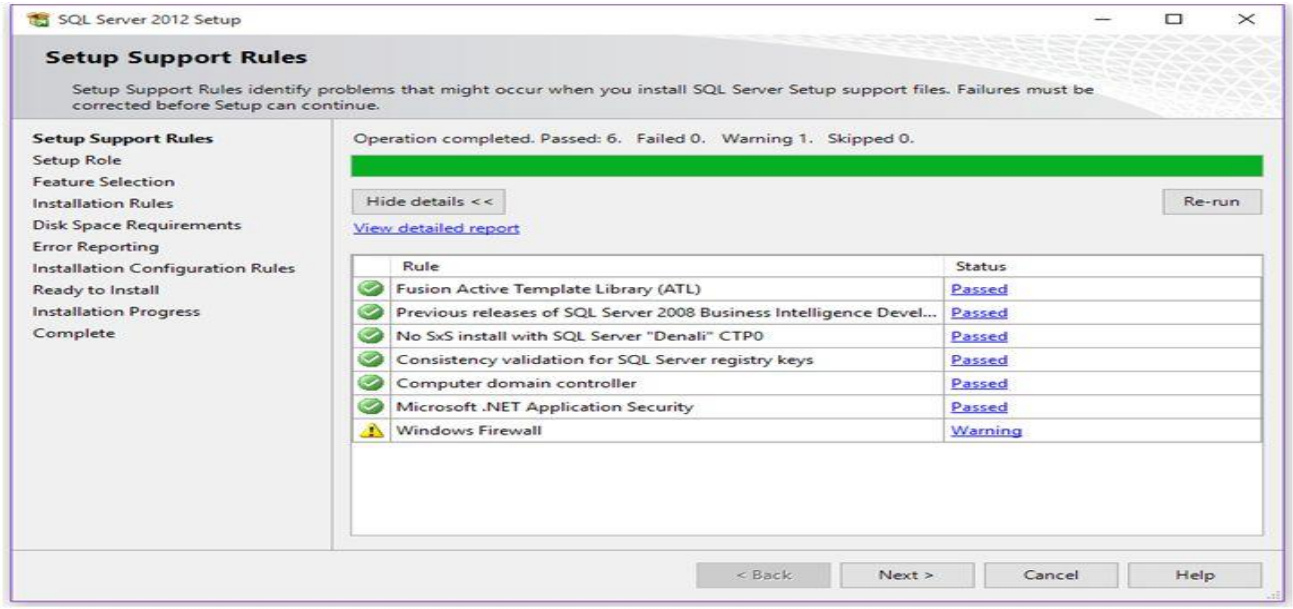
Step 7: Install Setup Files

At this window, you'll get Install button to install the updates.



Step 8: Setup Support Rules

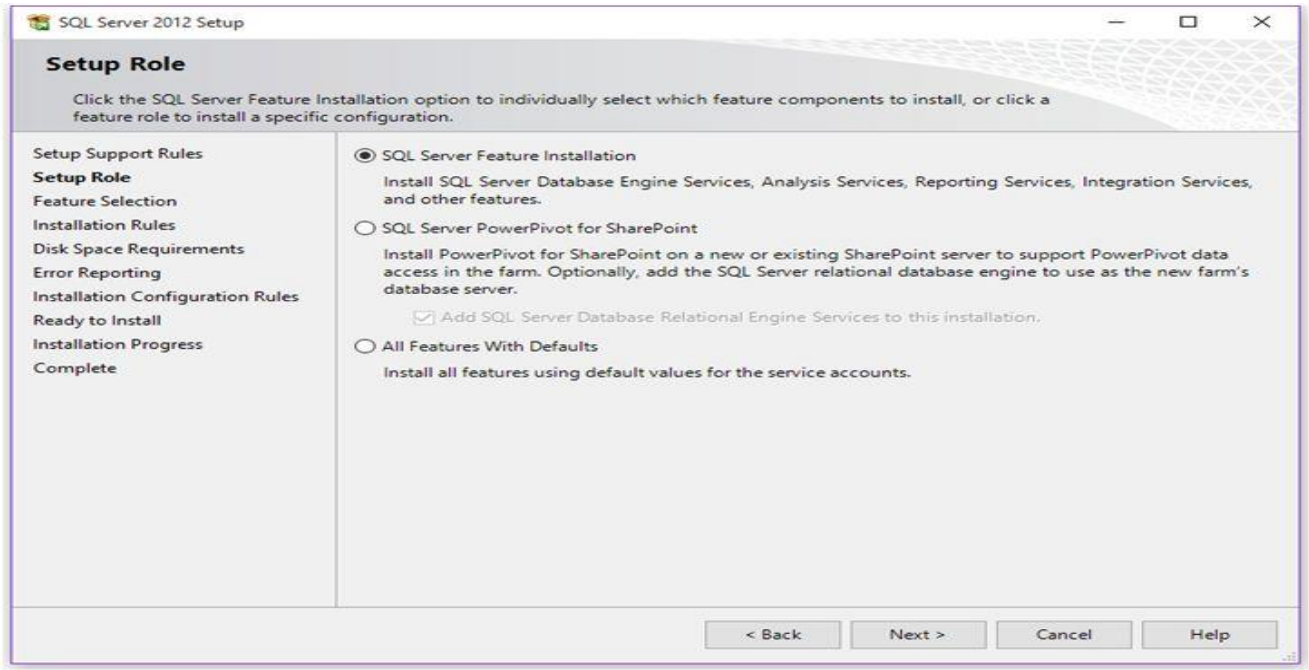
After successful completion of previous step, setup will again run a check to ensure everything looks good for the installation.



Step

Step 9: Setup Role

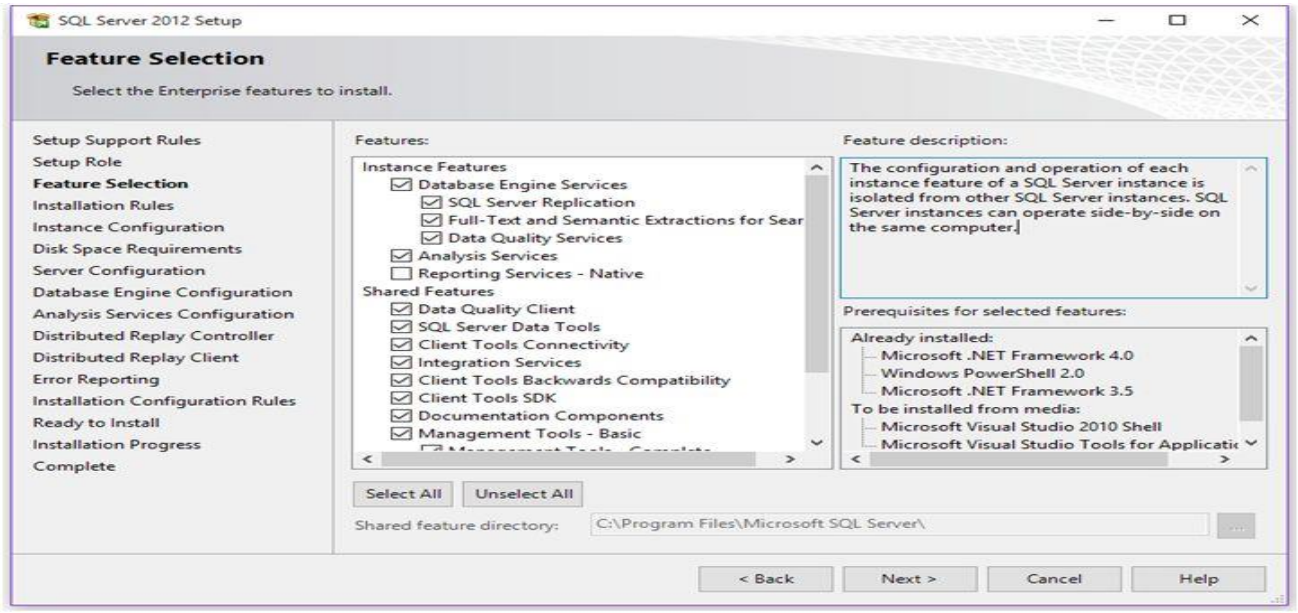
At this step, you'll find options like install SQL Server instance or install instance of Analysis Service with SharePoint integration. By default it'll select 'SQL Server Feature Installation'.



If you select "All Features with Default", the following things will be set by default:

- On the Feature Selection page, all features will be selected by default.
- On Server Configuration page, default accounts will be set.
- On Database Engine Configuration page, your current logon account will be added as a Server Administrator.

Step 10: Components or Features to Install



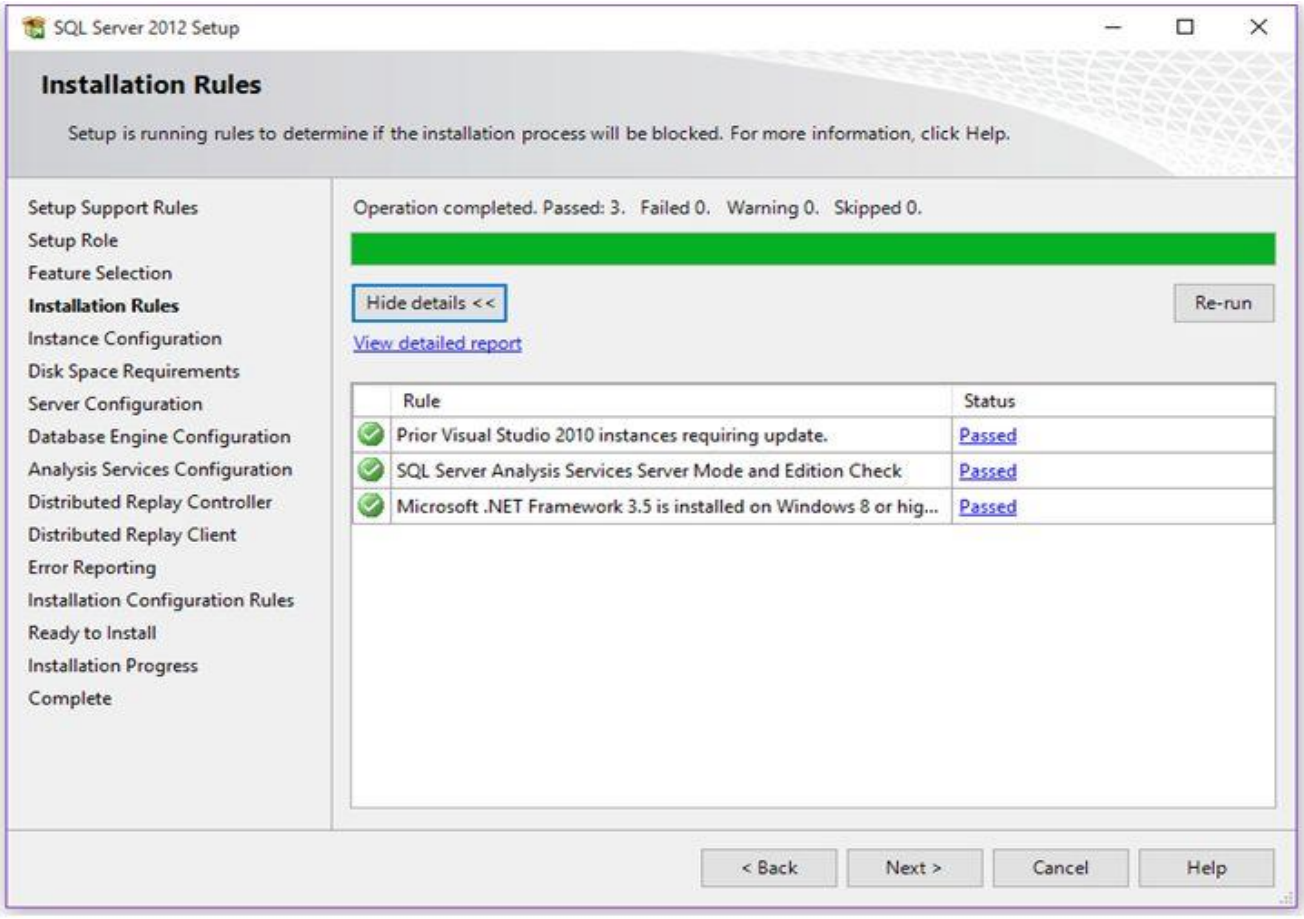
Select the components you want to install on your machine. The following is the description for above listed components.

- Database Engine Services: Allow you to install SQL Server instance.
- Analysis Services: Allow you to install an Analysis Services instance on standalone or on cluster node.
- Reporting Services: Allow you to install the server as report server.
- SQL Server Data Tool: Allow you to install SQL Server Developer tool to work with integration packages. In SQL Server 2008 installation you'll find this service named as 'Business Intelligence Development Studio'.
- Integration Services: Allow you to install Integration Services.
- Management Tool: Allow you to install SQL Server management configuration tool including command line and power shell tool.

If you selected “All Features with Default” in previous step, all these components will be checked automatically.

Step 11: Installation Rules

After selecting the features to install, setup again runs a check to ensure whether your machine’s configuration is compatible or not to proceed further.

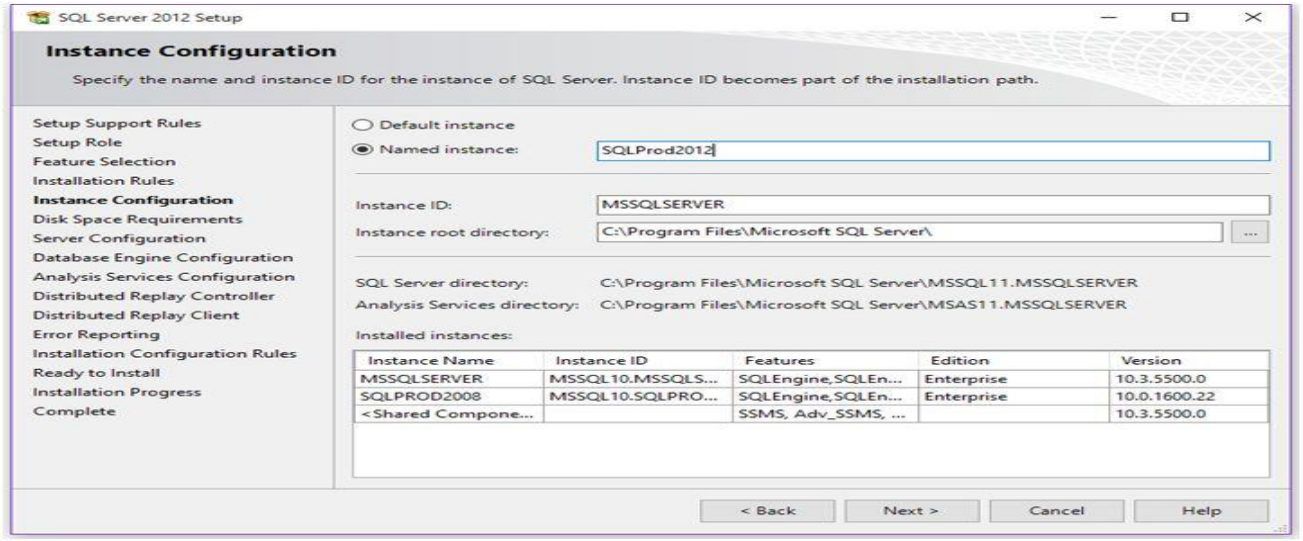


If all looks good, click Next.

Step 12: Instance Configuration

This step will ask, what type of instance you want to configure, as we all know, either we can install Default or Named instance. If default instance is already installed, you’ll have to have proceeded with

named instance.

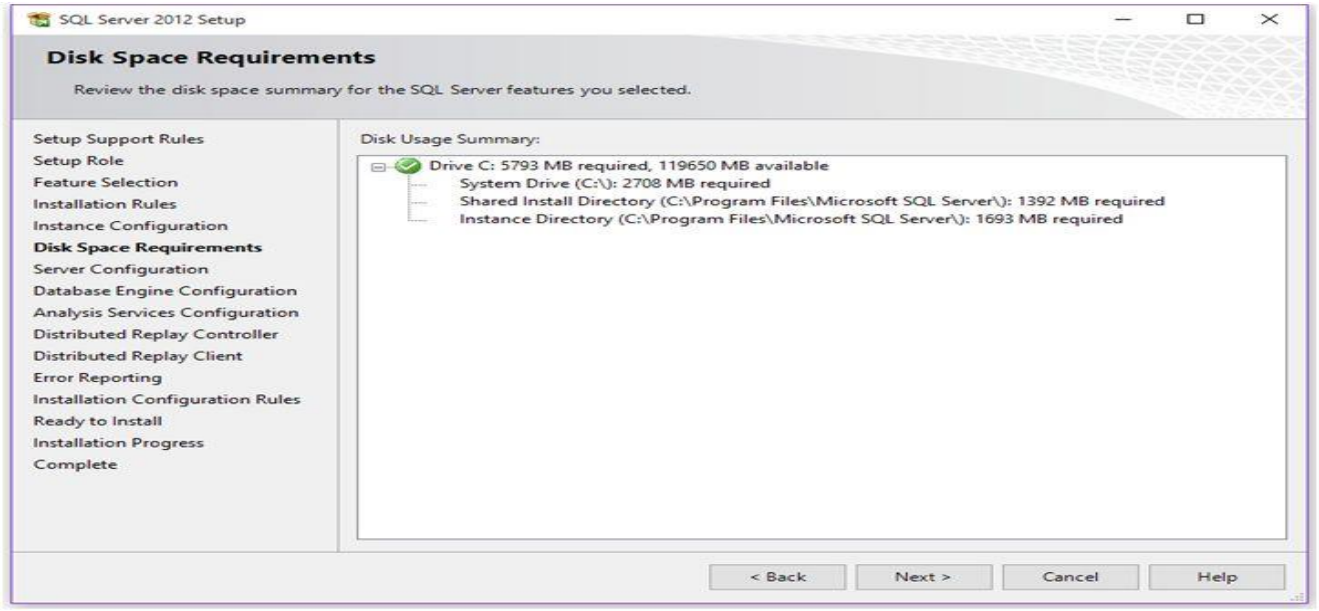


Select type of instance you want to install. If you want to change the root directory of your instance, you can change it from ‘Instance root directory’ option. It’ll also show you the instance already installed on your machine. As shown in above screenshot, I already installed 2 instances on my machine.

After doing instance configuration, click Next.

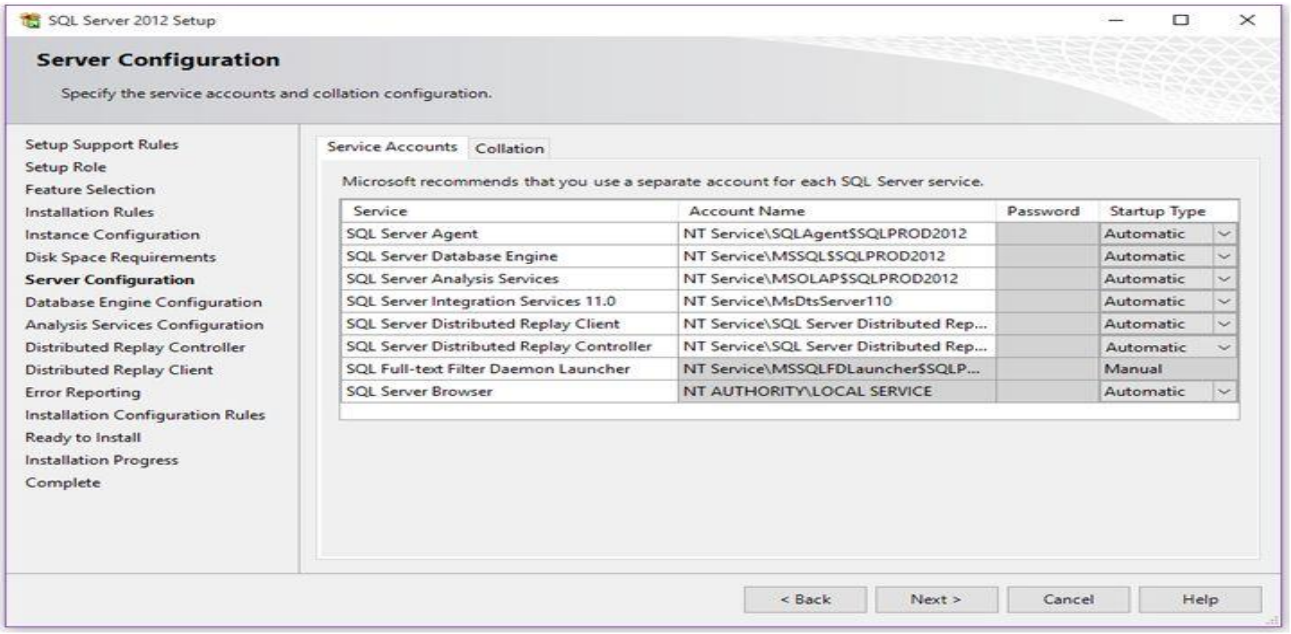
Step 13: Disk Space requirement summary

At this step, you’ll get disk space summary which will show how much disk space your instance will take on the machine.



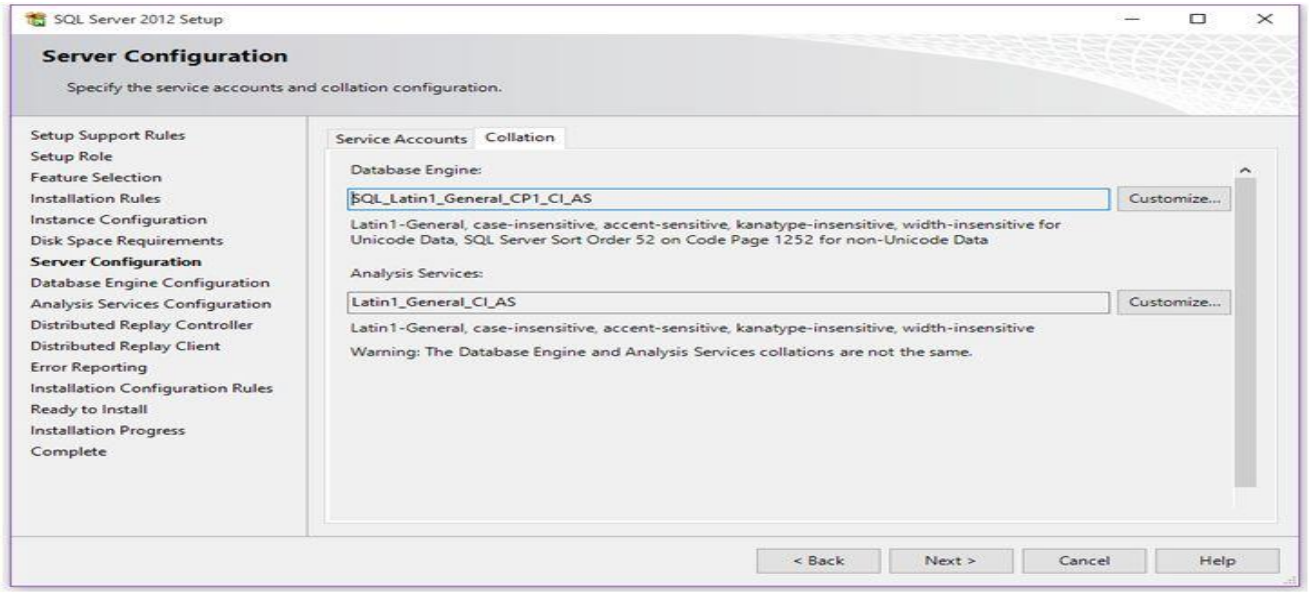
Step 14: Server Configuration

On this step you'll find options to specify Service Accounts and Collation Configuration.



Under Service Account tab, you'll find option to set account name and password for any of the services; also you can choose start-up type of those services. You can set startup type as Manual or Automatic. It's recommended to set Start-up type of SQL Services to Automatic.

Next, you'll find Collation tab where you can set collation level for SQL Server and Analysis Services. The collation type you select here will be set as default collation for your instance.

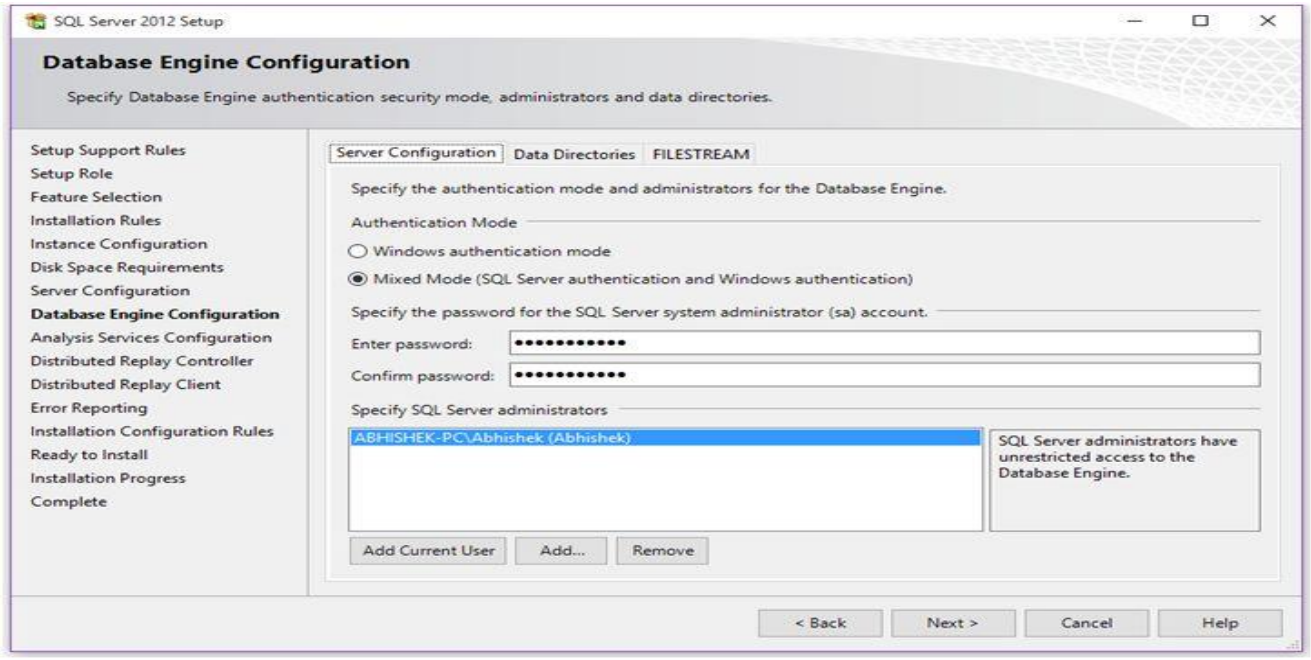


After performing above steps, click Next.

Step 15: Database Engine Configuration

This is the most important step because over here you'll configure your servers configuration, data directories and file stream options.

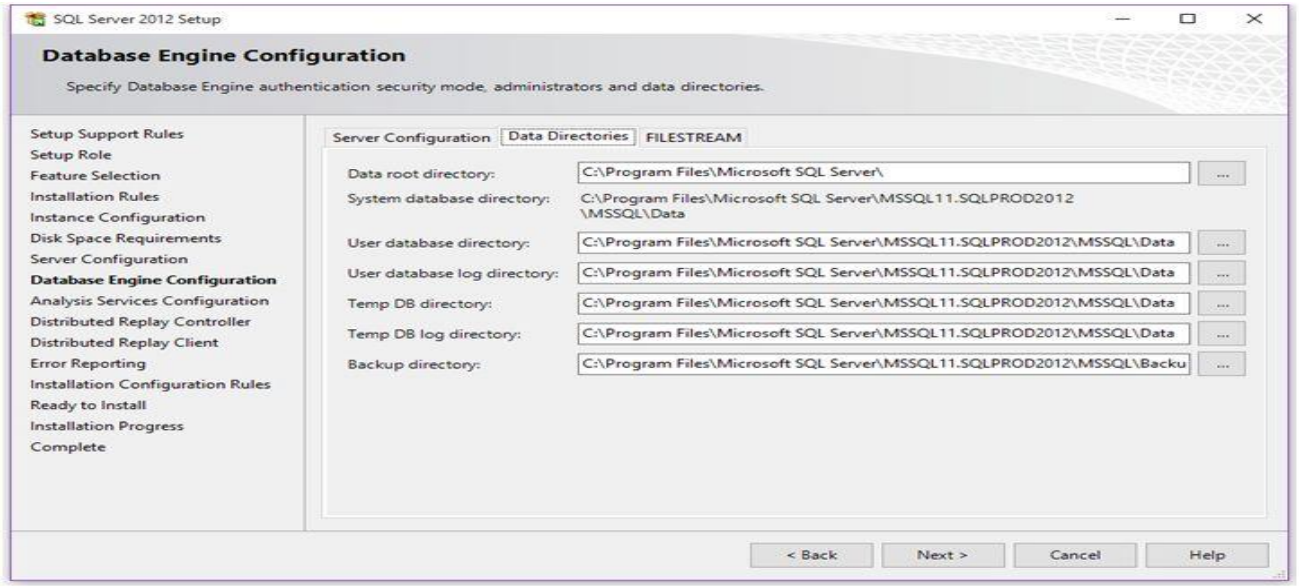
At Server Configuration tab, you'll find authentication mode and SQL Server System Administrator [SA] account configuration.



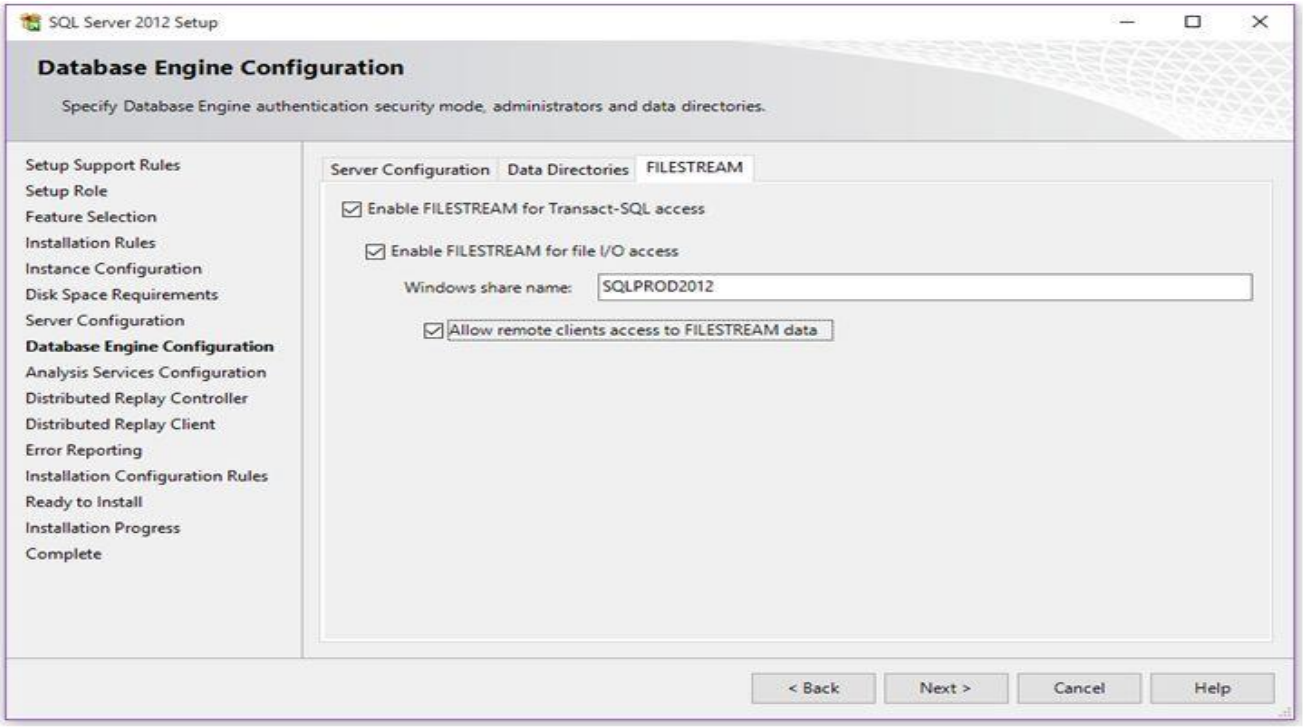
If you see Data Directories tab, you'll find your root directory, and location of below:

- Data root directory.
- User database directory,
- System database directory,
- User database log directory,
- Temp data and log directories, and
- Backup location

You can change these locations of your own choice.



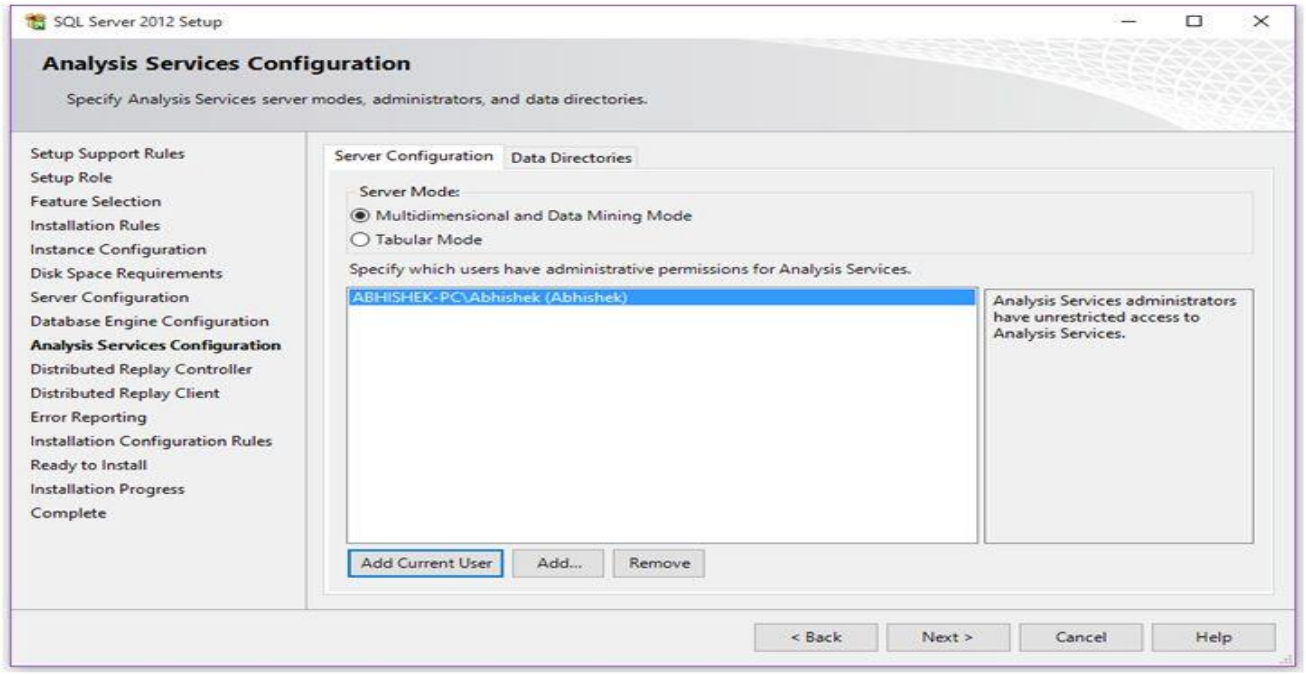
On File Stream tab, you'll find option to enable File Stream feature.



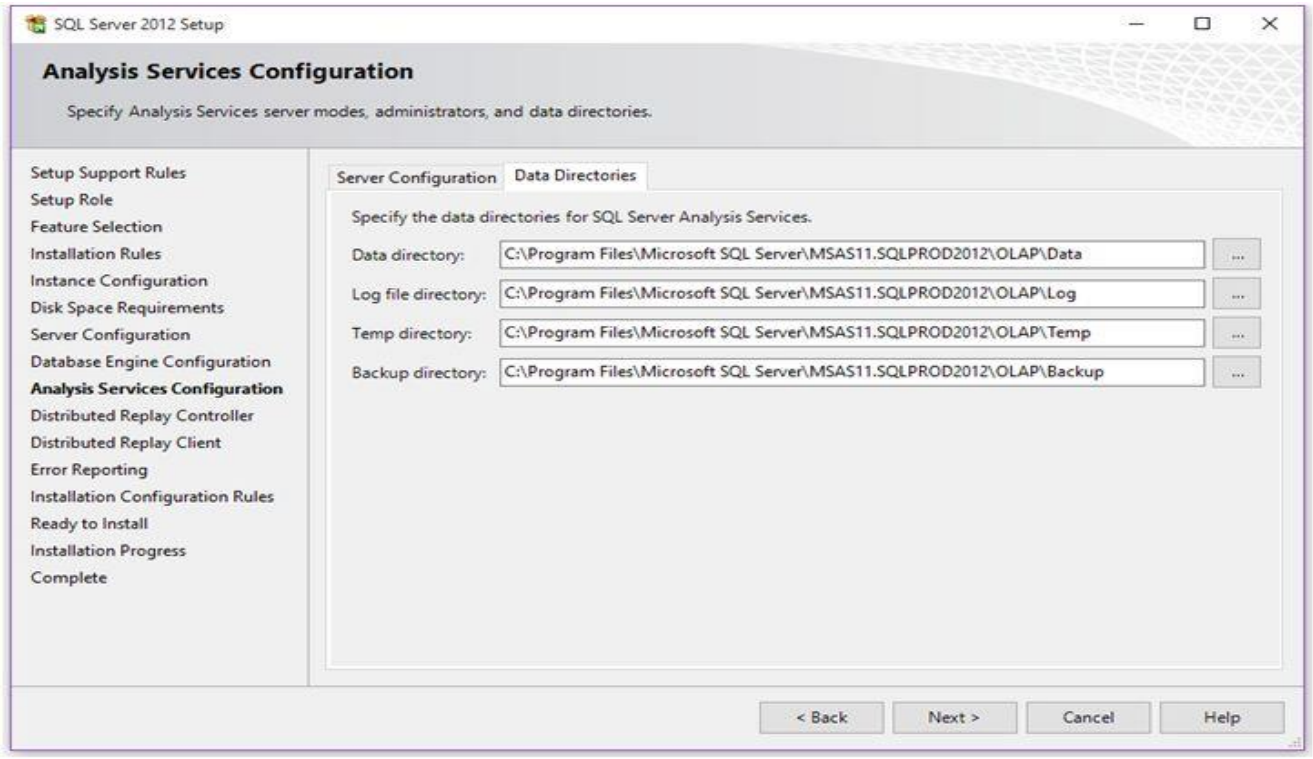
After completing the steps, click Next.

Step 16: Analysis Services Configuration

If you had chosen Analysis Services to be installed on your machine, you'll find this option during installation process. Complete this step by choosing server mode and adding Analysis Services Administrator.



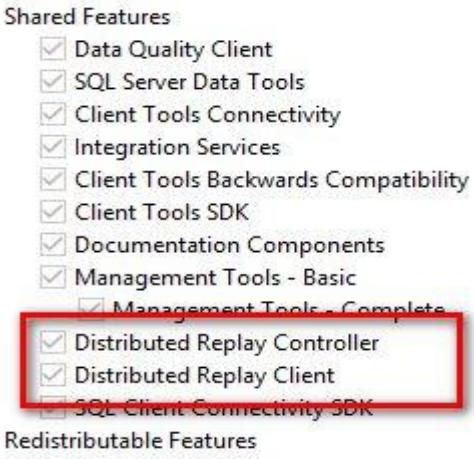
Specify your Analysis Services data directories as shown below.



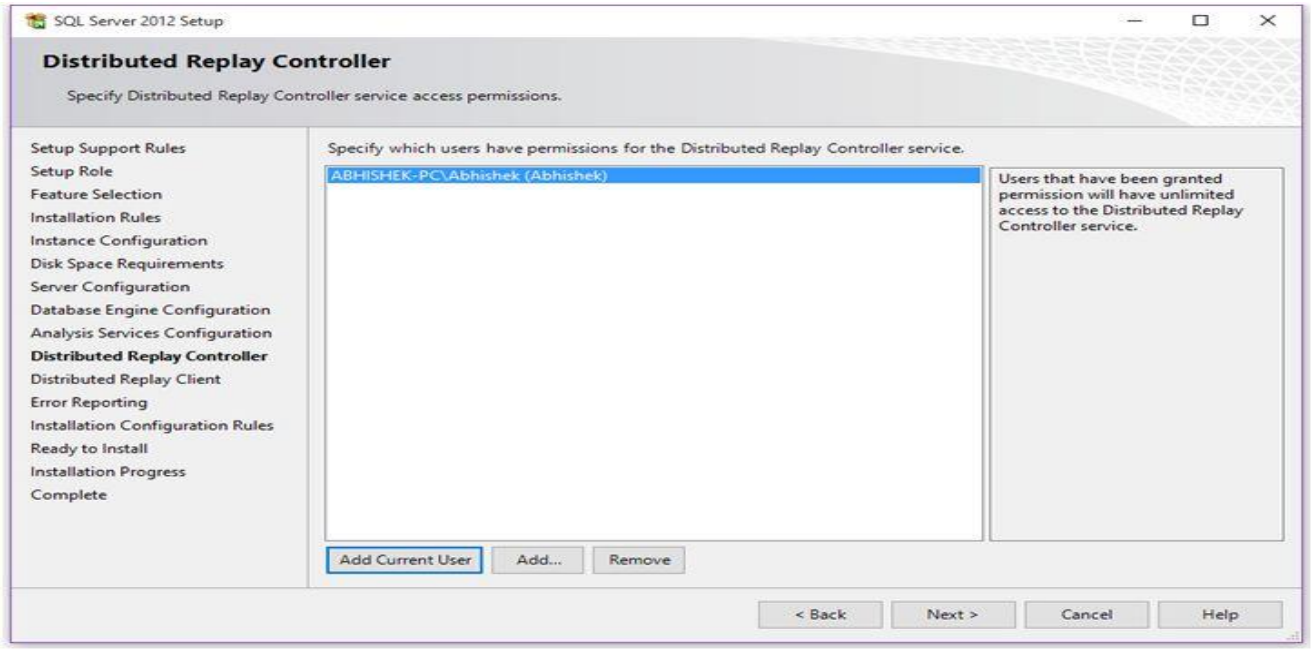
Click Next,

Step 17: Distributed Replay Controller

If you had selected all shared features to install, you'll get this and next step to complete.



At this step, you'll find Distributed Replay Controller. This feature helps you assess the impact of future SQL Server upgrades.

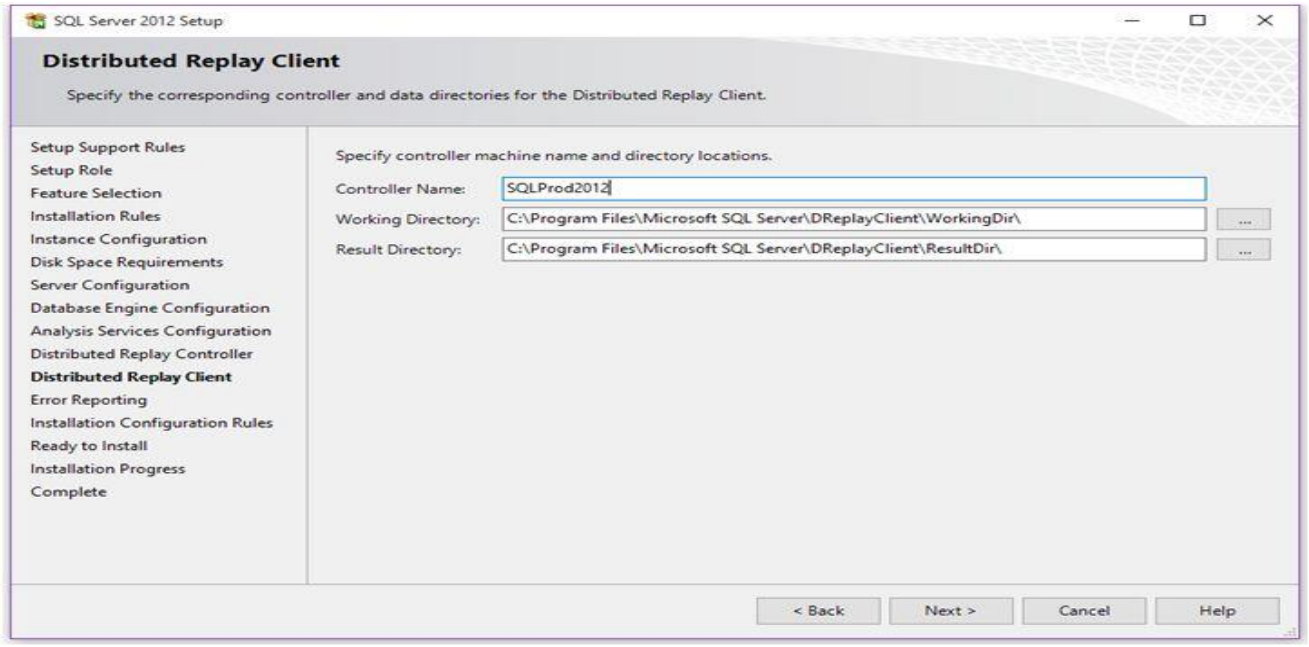


This is similar to SQL Server Profile, Distributed Replay Controller replay a captured trace against an upgraded test environment. This feature can use multiple computers to replay trace data and simulate a mission-critical workload.

Add users to have unlimited access to the Distributed Replay client service and click next.

Step 18: Distributed Replay Client

This is one of the component of Distributed Replay Controller under which one or more computers (physical or virtual) running the Windows service named SQL Server Distributed Replay client. The Distributed Replay client works together to simulate workloads against an instance of SQL Server.



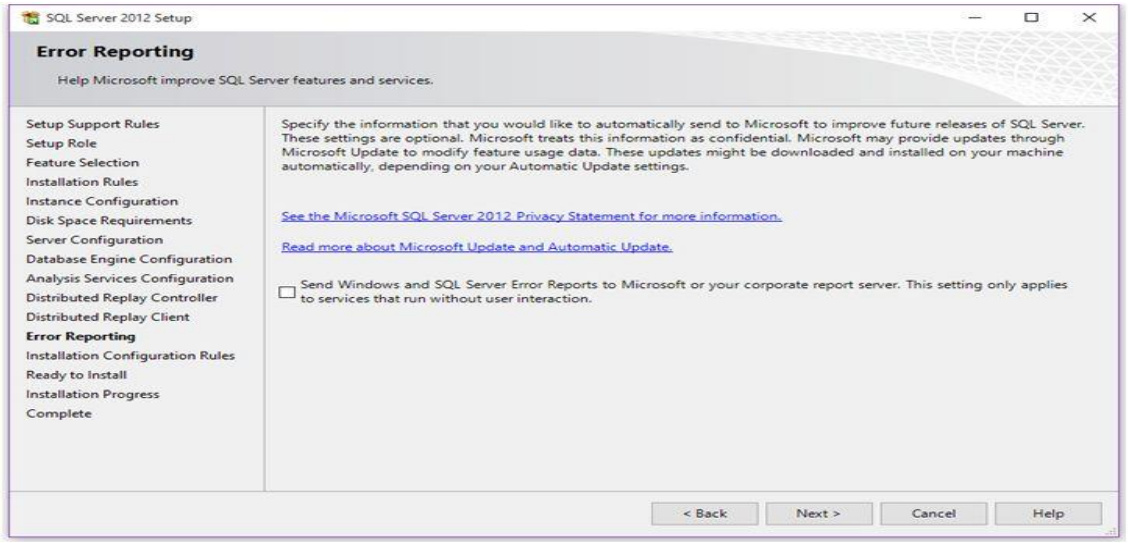
Enter Controller Name so that the client computer will communicate with for the Distributed Replay Client Service. This is the optional parameter and the default value is ‘BLANK’.

Specify the working directory for the Distributed Replay client service.

Specify the result directory for the Distributed Replay client service.

Click Next.

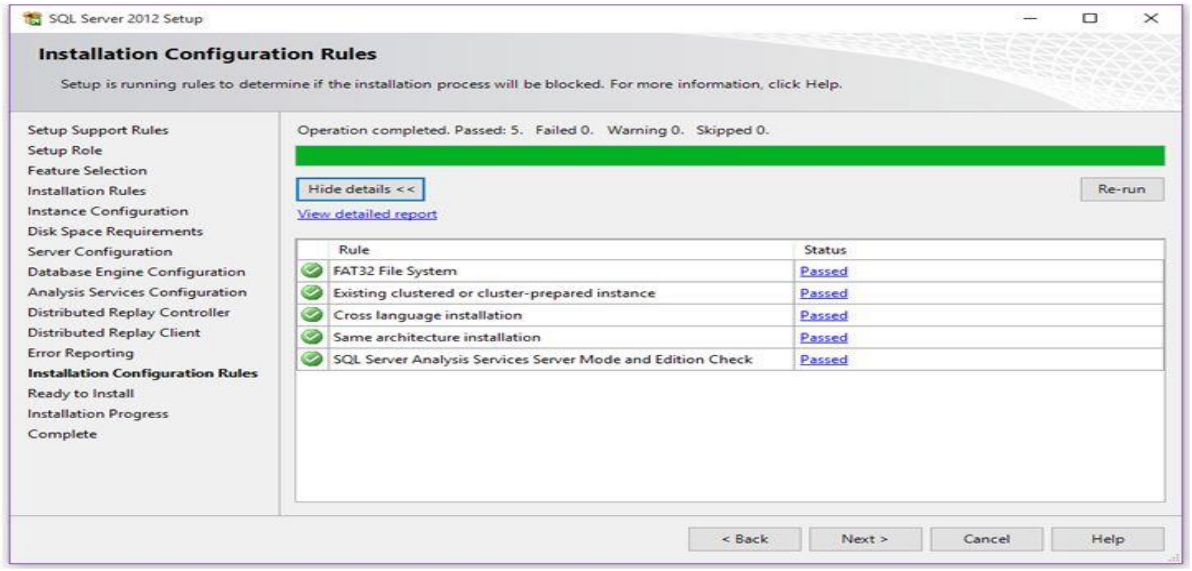
Step 19: Error Reporting to Microsoft.



Click Next.

Step 20: Installation Configuration Rule

At this step, setup will perform a final check to ensure everything looks good for installation operation.



If all rule passes, click Next.

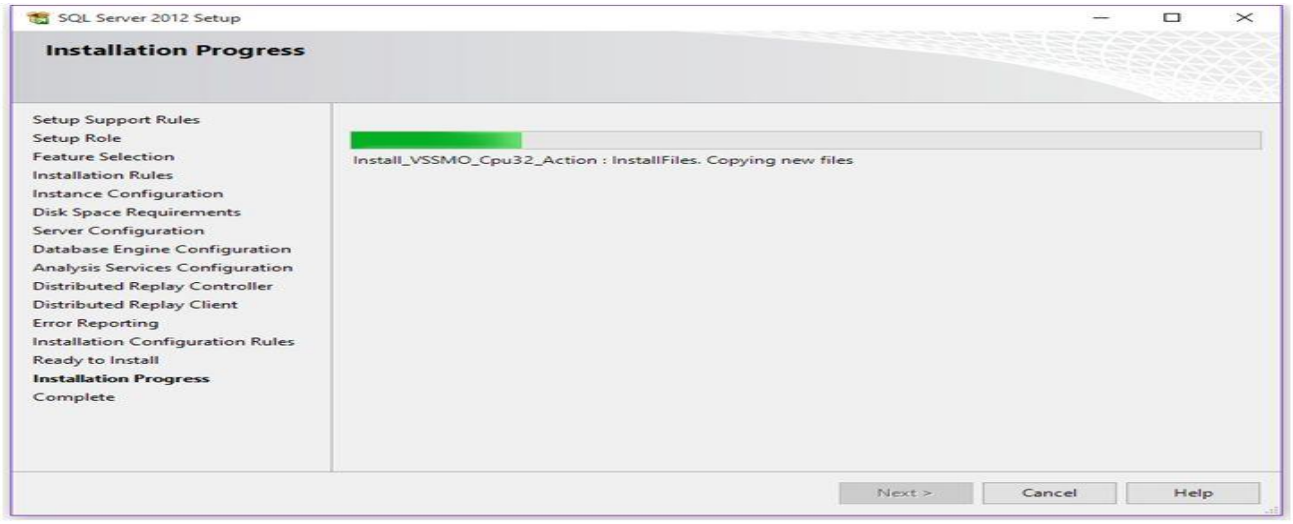
Step 21: Installation Summary

Here you'll get summary of your installation as shown below.

If you're satisfied with everything, click on Install button and you're set to go.

Step 22: Ready Steady Go!

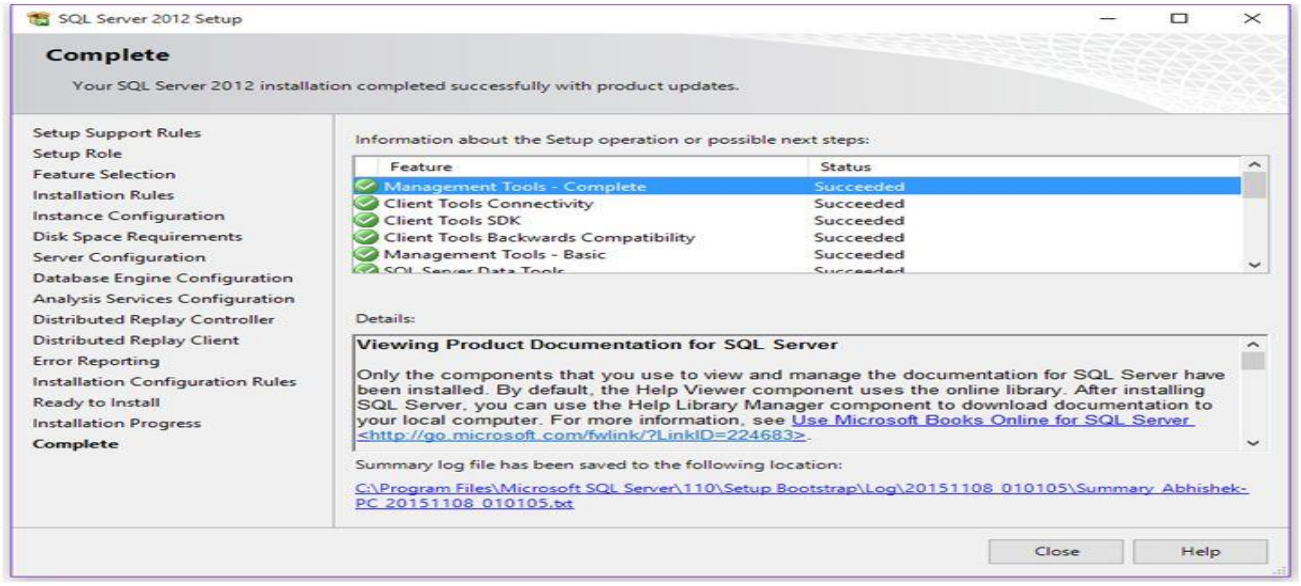
Installation process will start and you'll see the progress as in the following,



This will take some time, relax and just watch the progress.

Step 23: Installation completed

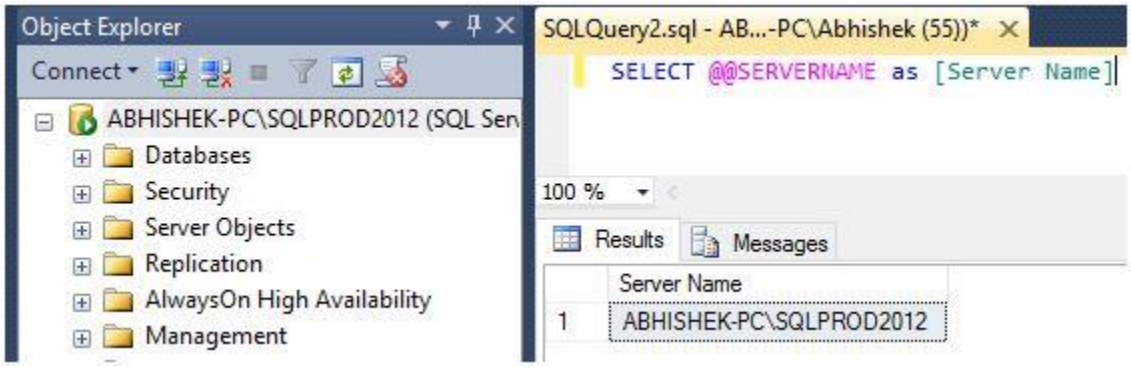
After successful installation you'll get the following window. This will show you the components installed on your machine with 'Succeeded' message in Status column.



Also you'll get location of the log file of the complete installation.

Step 24: Verification

Open SSMS and connect your instance and you're ready to explore.



Conclusion

We've successfully installed SQL Server 2012 named instance on our machine. We have seen the steps involved in the installation.

Lap Test

Instruction: Given necessary templates, tools and materials you are required to perform the following tasks accordingly

Task 1: Connect your SQL server

Task 2: Create a new query

Unit Two: Data definition language

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

- Introduction to SQL data definition language commands
- Database planning
- Naming convention for all database elements
- Database structure creation and manipulation

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:

- Explain Data definition language
- Identify DDL commands
- Apply SQL DDL to create and manipulate database structures
- Identify the key components of a relational database (tables, rows, columns).
- Understand the concept of table relationships (e.g., one-to-many, many-to-many).
- Utilize INNER JOIN, LEFT JOIN, RIGHT JOIN, and FULL JOIN to combine data from multiple tables.
- Define and enforce data integrity using constraints (e.g., PRIMARY KEY, FOREIGN KEY, UNIQUE).
- Understand the purpose of constraints in maintaining data quality.
- Understand the purpose and benefits of creating views

2.1. Introduction to SQL data definition language commands

The Data Definition Language (DDL) is part of SQL that you use to create (completely define) a database, modify its structure, and destroy it when you no longer need it.
It contains SQL commands you use to create, change, or destroy the basic elements of a relational database. Basic elements include tables, views, schemas, catalogs, clusters, indexes, stored procedures, functions and possibly other things as well. In this section, you will see containment hierarchy that
relates these elements to each other and look at the commands that operate on these elements. Schema is an overall structure that includes tables within it. Tables and schemas are two elements of a relational database's containment hierarchy. You can break down the containment hierarchy as follows:

- Tables contain columns and rows.
- Schemas contain tables and views.
- Catalogs contain schemas.

The database itself contains catalogs. Sometimes the database is referred to as a cluster.

A database table is a two-dimensional array made up of rows and columns. You can create a table by using the SQL CREATE TABLE command. Within the command, you specify the name and data type of each column. After you create a table, you can start loading it with data. (Loading data is a DML, not a DDL, function.). If requirements change, you can change a table's structure by using the ALTER TABLE command. If a table outlives its usefulness or becomes obsolete, you can eliminate it with the DROP command. The various forms of the CREATE and ALTER commands, together with the DROP command, make up SQL's DDL.

2.2. Database planning

Say that you need to create a database for your organization. Excited by the prospect of building a useful, valuable, and totally righteous structure of great importance to your company's future, you sit down at your computer and start entering SQL CREATE commands. Right? Well, no. Not quite. In fact, that's a prescription for disaster. Many database development projects go awry from the start as excitement and enthusiasm overtake careful planning. Even if you have a clear idea of how to structure

your database, write everything down on paper before touching your keyboard. Keep in mind the following procedures when planning your database:

- Identify all tables.
- Define the columns that each table must contain.
- Give each table a primary key that you can guarantee is unique.
- Make sure that every table in the database has at least one column in common with one other table in the database. These shared columns serve as logical links that enable you to relate information in one table to the corresponding information in another table.
- Put each table in third normal form (3NF) or better to ensure the prevention of insertion, deletion, and update anomalies. After you complete the design on paper and verify that it is sound, you're ready to transfer the design to the computer by using SQL CREATE commands.

2.3. Usage of relevant naming convention for all database elements

The objects in a SQL-based database are identified by assigning them unique names. Names are used in SQL statements to identify the database object on which the statement should act.

The ANSI/ISO SQL standard specifies tables names (which identify tables), column names (which identify columns), and user names (which identify users of the database)

The SQL database Name should not be empty and special characters. The ANSI/ISO standards specifies that SQL names must contain 1 to 18 characters, begin with a letter, and my not contain any spaces or special punctuation characters.

- **Table Name**

When you specify a table name in a SQL statement, SQL assumes that you are referring to one of your own tables (that is, a table that you created). With the proper permission, you can also refer to tables owned by other users using a ***qualified table name***.

- **Column Name**

When you specify a column name in a SQL statement, SQL can normally determine from the context which column you intend. A column name should not be blank and the same in the same database.

- **Data Types**

There is a standard that specifies various types of data that can be stored in SQL-based database and manipulated by the SQL languages.

Character strings:

Data type	Description	Storage
char(n) or character(n)	Fixed-length character string. Maximum 8,000 characters	N
varchar(n)	Variable-length character string. Maximum 8,000 characters	
varchar(max)	Variable-length character string. Maximum 1,073,741,824 characters	
Text	Variable-length character string. Maximum 2GB of text data	

- **Unicode strings:**

Data type	Description	Storage
nchar(n)	Fixed-length Unicode data. Maximum 4,000 characters	
nvarchar(n)	Variable-length Unicode data. Maximum 4,000 characters	
nvarchar(max)	Variable-length Unicode data. Maximum 536,870,912 characters	

Ntext	Variable-length Unicode data. Maximum 2GB of text data	

- **Binary types:**

Data type	Description	Storage
Bit	Allows 0, 1, or NULL	
binary(n)	Fixed-length binary data. Maximum 8,000 bytes	
varbinary(n)	Variable-length binary data. Maximum 8,000 bytes	
varbinary(max)	Variable-length binary data. Maximum 2GB	
Image	Variable-length binary data. Maximum 2GB	

- **Number types:**

Data type	Description	Storage
tinyint	Allows whole numbers from 0 to 255	1 byte
smallint	Allows whole numbers between -32,768 and 32,767	2 bytes
Int	Allows whole numbers between -2,147,483,648 and 2,147,483,647	4 bytes
Bigint	Allows whole numbers between -9,223,372,036,854,775,808 and 9,223,372,036,854,775,807	8 bytes
decimal(p,s)	Fixed precision and scale numbers. Allows numbers from -10^38 +1 to 10^38 –1. The p parameter indicates the maximum total number of digits that can be stored (both to the left and to the right of the decimal point). p must be a value from 1 to 38. Default is 18. The s parameter indicates the maximum number of digits stored to	5-17 bytes

	the right of the decimal point. s must be a value from 0 to p. Default value is 0	
numeric(p,s)	Fixed precision and scale numbers. Allows numbers from -10^38 +1 to 10^38 –1. The p parameter indicates the maximum total number of digits that can be stored (both to the left and to the right of the decimal point). p must be a value from 1 to 38. Default is 18. The s parameter indicates the maximum number of digits stored to the right of the decimal point. s must be a value from 0 to p. Default value is 0	5-17 bytes
smallmoney	Monetary data from -214,748.3648 to 214,748.3647	4 bytes
money	Monetary data from -922,337,203,685,477.5808 to 922,337,203,685,477.5807	8 bytes
float(n)	Floating precision number data from -1.79E + 308 to 1.79E + 308. The n parameter indicates whether the field should hold 4 or 8 bytes. float(24) holds a 4-byte field and float(53) holds an 8-byte field. Default value of n is 53.	4 or 8 bytes
Real	Floating precision number data from -3.40E + 38 to 3.40E + 38	4 bytes

- **Date types:**

Data type	Description	Storage
datetime	From January 1, 1753 to December 31, 9999 with an accuracy of 3.33 milliseconds	8 bytes
datetime2	From January 1, 0001 and December 31, 9999 with an accuracy of 100 nanoseconds	6-8 bytes
smalldatetime	From January 1, 1900 to June 6, 2079 with an accuracy of 1 minute	4 bytes
Date	Store a date only. From January 1, 0001 to December 31, 9999	3 bytes
Time	Store a time only to an accuracy of 100 nanoseconds	3-5 bytes
datetimeoffset	The same as datetime2 with the addition of a time zone offset	8-10 bytes
timestamp	Stores a unique number that gets updated every time a row gets created or modified. The timestamp value is based upon an internal clock and does not correspond to real time. Each table may have only one timestamp variable	

- **Other data types:**

Data type	Description
sql_variant	Stores up to 8,000 bytes of data of various data types, except text, ntext, and timestamp
uniqueidentifier	Stores a globally unique identifier (GUID)
Xml	Stores XML formatted data. Maximum 2GB
cursor	Stores a reference to a cursor used for database operations
Table	Stores a result-set for later processing

- ***Fixed-length character strings***: columns holding these types of data typically store names of people and companies, addresses, descriptions, and so on.
- ***Integers:*** columns holding this types of data typically store counts, quantities, ages, and so on. Integer's columns are also frequently used to contain Id numbers, such as customers, employee. And order numbers.
- ***Decimal numbers***: columns with this type store numbers that have fractional parts and must be calculated exactly, such as rates and percentages. They are also frequently used to store money amounts.

- **Extended data Types**

- **Variable-length Character string:** SQL which supports VARCHAT data. Which allows a column to store character strings that vary in length from row to row, up to some maximum length.
- **Dates and times:** supports for date/time values.
- **Boolean data:** supports logical(TRUE or FALSE) values as an explicit type.

- **Data Types Differences**

The differences b/n the data types offered in various SQL implementation is one of the practical barriers to the portability of SQL based applications.

Example: Date/time data provides an excellent example of theses differences.

Date: w/c stores a date like ***June 30, 2009*** or *30 June 2009*

- **Constants**

In some SQL statements a numeric, character, or date data value must be expressed in text form.

For example: INSERT statement, w/c adds a student to the database:

```
INSERT INTO student(Fname, SID, Dept,
year) VALUES ('Alemayehu', 200, 'Comp
```

The value for each column in the newly inserted row is specified in the VALUES clause. Constant data values are also used in expression such as in the SELECT statement

```
SELECT
city FROM
offices
```

➢ **Numeric constant**

Integers and decimal constants (also called exact numeric literals) are written as ordinary decimal numbers in SQL statements, with an optional leading plus or minus sign

Example: 200 +345.95 -500 789.00

Use a comma between the digits of a numeric constant

- **String Constant**

The ANSI/ISO standard specifies that SQL constants for character data be enclosed in **single quotes** (**'……'**)

Example: 'Waksum Motuma' 'Addis Ababa'

If a single quotes is to be used included in the constant text, it is written within the constant as two consecutive single quote characters. This is constant value:

Example: "I can't"

- **Date and time constants**

In SQL products the supports date/time data. Constant values for dates, times, and time intervals are specified as ***string constants***. The format of these constants varies from one DBMS to the next.

Example:

```
SELECT    Name,    dept
FROM  student

WHERE  hire-date= To-date('June 30, 2009', 'mon DD yyyy')
```

- **Expressions**

Expressions are used in the SQL Languages to calculate values that are retrieved from the database and to calculate values used in searching the database.

Example1: The query that calculates the sales of each offices as a percentage of its target:

```
SELECT   city,  target,  sales,(sales/target)*100
FROM  offices
```

Example2:

```
SELECT        city
FROM offices
WHERE  sales > target +50000.00
```

- **Missing data (Null Values)**

It is a value whose value is missing, unknown, or don't apply.

SQL supports missing, unknown, or inapplicable data explicitly through the concepts of ***null values***.

A null Value is an indicator that tells SQL (and the user) that the data is missing or not applicable.

Example: given a student table below that has missing values

Name	IDNo	Sex	Age
Ayantu Hangassa	GSR/3456/99	M	32
Robe Tasisa	GRS/5690/99	F	Null

Value unknown

2.4. Database structure creation and manipulation

Remember that a database is designed, built and populated with data for a specific purpose that is designed to address a specific problem in the real world.

Once you have completed database design, you can implement it using a specific DBMS. You may, for example, create a CUSTOMER table with the attributes CUSTOMER.CustomerID, CUSTOMER.FirstName, CUSTOMER.LastName, CUSTOMER.Street, CUSTOMER.City, CUSTOMER.State, CUSTOMER.Zipcode, and CUSTOMER.Phone. All of these attributes are more

closely related to the customer entity than to any other entity in a database that may contain many tables. These attributes contain all the relatively permanent customer information that your organization keeps on file. Most database management systems provide a graphical tool for creating database tables. You use SQL Server management studio which is a graphical tool provided by Microsoft to connect to MS SQL Server for creating database tables You can also create such tables by using an SQL command. Since a database contains all other objects including tables, you need to create database first before you create tables and other objects.

The SQL CREATE DATABASE statement is used to create new SQL database.

Syntax:

- Basic syntax of CREATE DATABASE statement is as follows:
  CREATE DATABASE DatabaseName;
  Always database name should be unique within the RDBMS.
  Example:

If you want to create new database <testDB>, then CREATE DATABASE statement would be as follows:
Eg. Create database testDB
You need to execute the command to create the specified database. Once you create a database and made it the active (currently selected) database, now you can create as many objects as you want including tables. The following example demonstrates a command that creates your CUSTOMER table:
CREATE TABLE CUSTOMER (customerid INTEGER NOT NULL, firstname CHARACTER (15), lastname CHARACTER (20) NOT NULL, Street CHARACTER (25), City CHARACTER (20), State CHARACTER (2), Zipcode INTEGER, Phone CHARACTER (13)) ;
For each column, you specify its name (for example, CustomerID), its data type (for example, INTEGER), and possibly one or more constraints (for example, NOT NULL).
View:- At times, you want to retrieve specific information from the CUSTOMER table.
You don't want to look at everything — only specific columns and rows. What you need is a view.
A view is a virtual table. In most implementations, a view has no independent physical existence. The view's definition exists only in the database's metadata, but the data comes from the table or tables from which you

derive the view. The view's data is not physically duplicated somewhere else in online disk storage. Some views consist of specific columns and rows of a single table. Others, known as multi table views, draw from two or more tables. Eg. Of SQL DDL statement to create a view

CREATE VIEW NH_CUST AS SELECT CUSTOMER.FirstName, CUSTOMER.LastName, CUSTOMER.Phone FROM CUSTOMER WHERE CUSTOMER.State = 'NH'

- **Relationships**

If one table in a database contains as a foreign key a column that is a primary key in another table in the database, you can add a constraint to the first table so that it references the second table.

Example:

CREATE TABLE CUSTOMERS(ID INT NOT NULL, NAME VARCHAR (20) NOT NULL, AGE INT NOT NULL, ADDRESS CHAR (25) , SALARY DECIMAL (18, 2), PRIMARY KEY (ID));

CREATE TABLE ORDERS (ID INT NOT NULL, O_DATE DATETIME, CUSTOMER_ID INT foreign key references CUSTOMERS(ID),AMOUNT Decimal(8,2), PRIMARY KEY (ID));

For defining a PRIMARY KEY constraint on multiple columns, use the following SQL syntax:

CREATE TABLE CUSTOMERS(ID INT NOT NULL, NAME VARCHAR (20) NOT NULL, AGE INT NOT NULL, ADDRESS CHAR (25), SALARY DECIMAL (18, 2), PRIMARY KEY (ID, NAME));

CHECK Constraint:

The CHECK Constraint enables a condition to check the value being entered into a record. If the condition evaluates to false, the record violates the constraint and isn't entered into the table.

Example:

For example, the following SQL creates a new table called CUSTOMERS and adds five columns. Here, we add a CHECK with AGE column, so that you can not have any CUSTOMER below 18 years:

CREATE TABLE CUSTOMERS(ID INT NOT NULL, NAME VARCHAR (20) NOT NULL, AGE INT NOT NULL CHECK (AGE >= 18), ADDRESS CHAR (25) , SALARY DECIMAL (18, 2), PRIMARY KEY (ID));

SQL NOT NULL Constraint

- By default, a table column can hold NULL values.
- The NOT NULL constraint enforces a column to NOT accept NULL values.
- The NOT NULL constraint enforces a field to always contain a value. This means that you cannot insert a new record, or update a record without adding a value to this field.
- The following SQL enforces the "P_Id" column and the "LastName" column to not accept NULL values:

```
CREATE TABLE Persons
(P_Id int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255))
```

SQL UNIQUE Constraint

- The UNIQUE constraint uniquely identifies each record in a database table.
- The UNIQUE and PRIMARY KEY constraints both provide a guarantee for uniqueness for a column or set of columns.
- A PRIMARY KEY constraint automatically has a UNIQUE constraint defined on it.
- Note that you can have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table.

SQL UNIQUE Constraint on CREATE TABLE

The following SQL creates a UNIQUE constraint on the "P_Id" column when the "Persons" table is created:

```
CREATE TABLE Persons (P_Id int NOT NULL UNIQUE,
LastName varchar(255) NOT NULL, FirstName varchar(255),
Address varchar(255),
City varchar(255))
```

To allow naming of a UNIQUE constraint, and for defining a UNIQUE constraint on multiple columns, use the following SQL syntax:

```
CREATE TABLE Persons (P_Id int NOT NULL,
LastName varchar(255) NOT NULL, FirstName varchar(255),
Address varchar(255), City varchar(255),
CONSTRAINT uc_PersonID UNIQUE (P_Id,LastName))
```

SQL DEFAULT Constraint

The DEFAULT constraint is used to insert a default value into a column.

The default value will be added to all new records, if no other value is specified.

The following SQL creates a DEFAULT constraint on the "City" column when the “Persons" table is created:

```
Create table persons(P_Id int NOT NULL, LastName varchar(255)
 NOT NULL, FirstName varchar(255), Address varchar(255),
City varchar(255) DEFAULT 'Sandnes')
```

The DEFAULT constraint can also be used to insert system values, by using functions like GETDATE():

```
CREATE TABLE Orders (O_Id int NOT NULL, OrderNo int NOT
NULL, P_Id int,OrderDate date DEFAULT GETDATE())
```

- **SQL AUTO INCREMENT Field**

Auto-increment allows a unique number to be generated when a new record is inserted into a table.

- **AUTO INCREMENT a Field**

Very often we would like the value of the primary key field to be created automatically every time a new record is inserted.

We would like to create an auto-increment field in a table.

Syntax for SQL Server

The following SQL statement defines the "P_Id" column to be an auto-increment primary key field in the "Persons" table:

```
CREATE TABLE Persons
(
P_Id int PRIMARY KEY IDENTITY,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255))
```

The MS SQL Server uses the ***IDENTITY*** keyword to perform an auto-increment feature.

By default, the starting value for IDENTITY is 1, and it will increment by 1 for each new record.

To specify that the "P_Id" column should start at value 10 and increment by 5, change the identity to IDENTITY(10,5).

To insert a new record into the "Persons" table, we will not have to specify a value for the "P_Id" column (a unique value will be added automatically):

```
INSERT INTO Persons (FirstName,LastName)
VALUES ('Lars','Monsen')
```

The SQL statement above would insert a new record into the "Persons" table. The "P_Id" column would be assigned a unique value. The "FirstName" column would be set to "Lars" and the "LastName" column would be set to "Monsen".

- **ALTER**

After you create a table, you're not necessarily stuck with that exact table forever. As you use the table, you may discover that it's not everything you need it to be. You can use the ALTER TABLE command to change the table by adding, changing, or deleting a column in the table. In addition to tables, you can also ALTER columns and domains. To create a PRIMARY KEY constraint on the "ID" column when CUSTOMERS table above already exists, use the following SQL syntax:

ALTER TABLE CUSTOMER ADD PRIMARY KEY (ID);

NOTE: If you use the ALTER TABLE statement to add a primary key, the primary key column(s) must already have been declared to not contain NULL values (when the table was first created).

If ORDERS table has already been created, and the foreign key has not yet been set, use the syntax for specifying a foreign key by altering a table. ALTER TABLE ORDERS ADD FOREIGN KEY (Customer_ID) REFERENCES CUSTOMERS (ID);

To create a PRIMARY KEY constraint on the "ID" and "NAMES" columns when CUSTOMERS table already exists, use the following SQL syntax:

ALTER TABLE CUSTOMERS ADD CONSTRAINT PK_CUSTID PRIMARY KEY (ID, NAME);

If ORDERS table has already been created, and the foreign key has not yet been set, use the syntax for specifying a foreign key by altering a table.

ALTER TABLE ORDERS ADD FOREIGN KEY (Customer_ID) REFERENCES CUSTOMERS (ID);

If CUSTOMERS table has already been created, then to add a CHECK constraint to AGE column, you would write a statement similar to the following:

ALTER TABLE CUSTOMERS ADD CONSTRAINT myCheckConstraint CHECK(AGE >= 18);

SQL UNIQUE Constraint on ALTER TABLE

To create a UNIQUE constraint on the "P_Id" column when the table is already created, use the following SQL:

```
ALTER TABLE Persons ADD UNIQUE (P_Id)
```

To allow naming of a UNIQUE constraint, and for defining a UNIQUE constraint on multiple columns, use the following SQL syntax:

```
ALTER TABLE Persons ADD CONSTRAINT uc_PersonID UNIQUE (P_Id,LastName)
```

- **SQL DEFAULT Constraint on CREATE TABLE**

To create a DEFAULT constraint on the "City" column when the table is already created, use the following SQL:

```
ALTER TABLE Persons
ALTER COLUMN City SET DEFAULT 'SANDNES'
```

- **DROP**

Removing a table from a database schema is easy. Just use a DROP TABLE <tablename>command. You erase all the table's data as well as the metadata that defines the table in the data dictionary. It's almost as if the table never existed. Eg Drop table Customers.

Delete Primary Key:-You can clear the primary key constraints from the table, Use Syntax:

ALTER TABLE CUSTOMERS DROP PRIMARY KEY PK_CUSTID;

To drop a FOREIGN KEY constraint, use the following SQL:

ALTER TABLE ORDERS DROP FOREIGN KEY Customer_ID;

To drop a UNIQUE constraint, use the following SQL:

```
ALTER TABLE Persons DROP CONSTRAINT uc_PersonID
```

To drop a DEFAULT constraint, use the following SQL:

```
ALTER TABLE Persons
ALTER COLUMN City DROP DEFAULT
```

DROP or DELETE Database

The SQL DROP DATABASE statement is used to drop an existing database in SQL schema.

Basic syntax of DROP DATABASE statement is as follows:

DROP DATABASE DatabaseName;

Example:

If you want to delete an existing database <testDB>, then DROP DATABASE statement would be as follows: Drop database testDB

The sporting goods store database contains four tables: CUSTOMER, PRODUCT, INVOICE, and INVOICE_LINE.

Exercise Write the SQL DDL command that implements the database. Name the database Sporting_goods.

Table	Column	Data type	Constraint
CUSTOMER	CustomerID	INTEGER	Primary key
	FirstName	CHARACTER (15)	
	LastName	CHARACTER(20)	NOT NULL
	Street	CHARACTER (25)	
	City	CHARACTER (20)	
	State	CHARACTER (2)	
	Zipcode	INTEGER	
	Phone	CHARACTER (13)	
PRODUCT	ProductID	INTEGER	Primary key
	Name	CHARACTER (25)	
	Description	CHARACTER (30)	
	Category	CHARACTER (15)	

	VendorID	INTEGER	
	VendorName	CHARACTER (30)	
INVOICE	InvoiceNumber	INTEGER	Primary key
	CustomerID	INTEGER	
	InvoiceDate	DATE	
INVOICE_LINE	LineNumber	INTEGER	Primary key
	InvoiceNumber	Integer	
	ProductID	INTEGER	
	Quantity	INTEGER	
	SalePrice	NUMERIC (9,2)	

Notice that some of the columns in the above Table contain the constraint primary key and NOT NULL. These columns are either the primary keys of their respective tables or columns that you decide must contain a value. A table's primary key must uniquely identify each row. To do that, the primary key must contain a non null value in every row. The tables relate to each other through the columns that they have in common.

The following list describes these relationships.

The CUSTOMER table bears a one-to-many relationship to the INVOICE table. One customer can make multiple purchases, generating multiple invoices. Each invoice, however, deals with one and only one customer.

The INVOICE table bears a one-to-many relationship to the INVOICE_LINEtable. An invoice may have multiple lines, but each line appears on one and only one invoice.

The PRODUCT table also bears a one-to-many relationship to the INVOICE_LINE table. A product may appear on more than one line on one or more invoices. Each line, however, deals with one, and only one product.

The CUSTOMER table links to the INVOICE table by the common CustomerID column. The INVOICE table links to the INVOICE_LINE table by the common InvoiceNumber column. The PRODUCT table links to the INVOICE_LINE table by the common ProductID column. These links are what makes this database a relational database.

Self-Check 2

Part I: Choose the correct answer from the alternatives provided

1. Which SQL statement is used to define a new table in a database?

A) SELECT B) CREATE C) UPDATE D) INSERT

2. What is the purpose of the SQL statement ALTER TABLE?

A) Insert new records into a table.

B) Delete existing records from a table.

C) Modify the structure of an existing table.

D) Retrieve data from one or more tables.

3. Which of the following SQL statements is used to remove a table from the database?

A) REMOVE TABLE

B) DROP TABLE

C) DELETE TABLE

D) ERASE TABLE

4. Which SQL data type is used to store variable-length character strings?

A) VARCHAR

B) INTEGER

C) FLOAT

D) DATE

Operation sheet 2.1: Create Database structure

Operation title: Creating database structure

Purpose: To create database structure using data definition language commands which will be used to store data.

Equipment and tools: Computer, DBMS software

Steps to create database structure

Step 1: Create a database named Sample_db

```
create database sample_db
use sample_db
```

Step 2: Create the following student table in your Sample_db database based on the information given below.

```
create table student(StudentID varchar(10) Primary key,Name char(30)Not null, Sex char(6)
Default 'Female' check( sex='male' or sex='Female'),BirtDate datetime Not null,Section char(6)
,DeptName char (40) Not null)
```

STUDENT

Field	Data type	size	Constraint
StudentID	varchar	10	Primary key
Name	char	30	Not null
Sex	char	6	Default “Female” , validate “male” or “Female”
BirtDate	datetime		Not null
Section	char	6	
DeptName	char	40	Not null

Step 3: Create the following Course table in your Sample_db database based on the information given below.

```
create table course (Course_Code varchar(8) Primary key ,Course_Title char (40) Not
null,Credit int )
```

COURSE

Field	Data type	size	Constraint
Course_Code	varchar	8	Primary key
Course_Title	char	40	Not null
Credit	int		

Step 3: Create the following Grade_Report table in your Sample_db database based on the information given below.

```
create table grade_report(StudentID Varchar (10)Foreign Key(StudentID)references
student,Course_Code Varchar (8) Foreign Key(Course_Code )references course, Grade
Char(1)check(grade='A'or grade= 'B' or grade='C'or grade='D' or grade= 'F'))
```

GRADE_REPORT

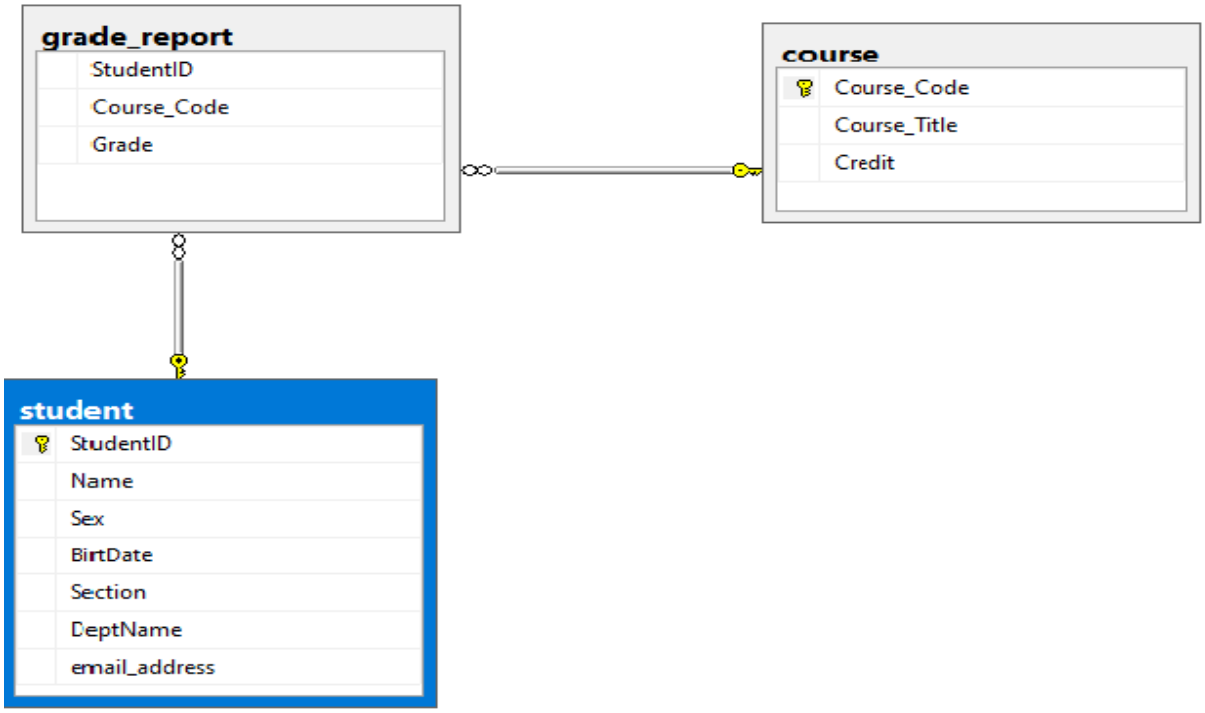
Field	Data type	size	Constraint
SID	Varchar	10	Primary key, Foreign Key
C_code	Varchar	8	Primary key, Foreign Key
Grade	char	1	Grade(A,B,C,D,F)

Step 4: Add new column into student table named “EmailAddress” with data type char and size 25.

```
alter table student add email_address char(25)
```

Step 5: Display a relationship you have created on step 3 above.

Click on the +sign of your instance name to expand it then expand the database node then click on your database name to expand it finally right click on database diagram and click new database diagram, click add to add all your diagram.


grade_report
StudentID
Course_Code
Grade
course
Course_Code
Course_Title
Credit
student
StudentID
Name
Sex
BirtDate
Section
DeptName
email_address

Lap Test

Instruction: Given necessary tools and materials you are required to perform the

following tasks accordingly

Task 1: Create a database called Hospital

Task 2: Create a table called Doctor, Patient and medicine with their respected relations

Task 3: Display a relationship you have created

Unit Three: Data manipulation language

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

- Overview of SQL data manipulation language commands
- Data insertion
- Modification of existing data
- Data deletion

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:

- Explain Data manipulation language
- Identify DML commands
- Perform data manipulation using INSERT, UPDATE, and DELETE statements.
- Understand the impact of these statements on the database.

3.1. Overview of SQL data manipulation language commands

The DDL is the part of SQL that creates, modifies, or destroys database structures; it doesn't deal with the data. The Data Manipulation Language (DML) is the part of SQL that operates on the data. Some DML statements read like ordinary English-language sentences and are easy to understand. Because SQL gives you very fine control of data, other DML statements can be fiendishly
complex. If a DML statement includes multiple expressions, clauses, predicates, or subqueries, understanding what that statement is trying to do can be a challenge.

3.2. Data insertion

SQL INSERT Query

The SQL INSERT INTO Statement is used to add new rows of data to a table in the database.

Syntax:

There are two basic syntaxes of INSERT INTO statement as follows:

INSERT INTO TABLE_NAME (column1, column2, column3,...columnN) VALUES (value1, value2, value3,...valueN);

Here, column1, column2,...columnN are the names of the columns in the table into which you want to insert data.

You may not need to specify the column(s) name in the SQL query if you are adding values for all the columns of the table. But make sure the order of the values is in the same order as the columns in the table. The SQL INSERT INTO syntax would be as follows:

INSERT INTO TABLE_NAME VALUES (value1,value2,value3,...valueN);

Example:

Following statements would create six records in CUSTOMERS table:

```
INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (1, 'Ramesh', 32, 'Ahmedabad', 2000.00 );
INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (2, 'Khilan', 25, 'Delhi', 1500.00 );
```

```
INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (3, 'kaushik', 23, 'Kota', 2000.00 );
INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (4, 'Chaitali', 25, 'Mumbai', 6500.00 );
INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (5, 'Hardik', 27, 'Bhopal', 8500.00 );
INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (6, 'Komal', 22, 'MP', 4500.00 );
```

You can create a record in CUSTOMERS table using second syntax as follows

```
INSERT INTO CUSTOMERS VALUES (7, 'Muffy', 24, 'Indore', 10000.00 );
```

All the above statements would produce the following records in CUSTOMERS table:

```
+----+----------+-----+-----------+----------+
| ID | NAME  | AGE | ADDRESS  | SALARY  |
+----+----------+-----+-----------+----------+
| 1 | Ramesh  | 32 | Ahmedabad |  2000.00 |
| 2 | Khilan  | 25 | Delhi  |  1500.00 |
| 3 | kaushik  | 23 | Kota  |  2000.00 |
| 4 | Chaitali |  25 | Mumbai  |  6500.00 |
| 5 | Hardik  |  27 | Bhopal  |  8500.00 |
| 6 | Komal  |  22 | MP  |  4500.00 |
| 7 | Muffy  |  24 | Indore  | 10000.00 |
+----+----------+-----+-----------+----------+
```

The following SQL statement will add a new row, but only add data in the "P_Id", "LastName" and the "FirstName" columns:

```
INSERT INTO Persons (P_Id, LastName, FirstName)
VALUES (5, 'Tjessem', 'Jakob')
```

The "Persons" table will now look like this:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger
4	Nilsen	Johan	Bakken 2	Stavanger
5	Tjessem	Jakob		

3.3. Modification of existing data

SQL UPDATE Query

The SQL UPDATE Query is used to modify the existing records in a table.

You can use WHERE clause with UPDATE query to update selected rows otherwise all the rows would be affected.

Syntax:

The basic syntax of UPDATE query with WHERE clause is as follows:

```
UPDATE table_name
SET column1 = value1, column2 = value2...., columnN = valueN
WHERE [condition];
```

You can combine N number of conditions using AND or OR operators.

Example:

Consider the CUSTOMERS table having the following records:

```
+----+----------+-----+-----------+----------+
| ID | NAME  | AGE | ADDRESS  | SALARY  |
+----+----------+-----+-----------+----------+
|  1 | Ramesh  |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan  |  25 | Delhi  |  1500.00 |
|  3 | kaushik  |  23 | Kota  |  2000.00 |
|  4 | Chaitali |  25 | Mumbai  |  6500.00 |
|  5 | Hardik  |  27 | Bhopal  |  8500.00 |
|  6 | Komal  |  22 | MP  |  4500.00 |
|  7 | Muffy  |  24 | Indore  | 10000.00 |
+----+----------+-----+-----------+----------+
```

Following is an example, which would update ADDRESS for a customer whose ID is 6:

```
UPDATE CUSTOMERS SET ADDRESS = 'Pune' WHERE ID = 6;
```

Now, CUSTOMERS table would have the following records:

```
+----+----------+-----+-----------+----------+
| ID | NAME  | AGE | ADDRESS  | SALARY  |
+----+----------+-----+-----------+----------+
|  1 | Ramesh  |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan  |  25 | Delhi  |  1500.00 |
|  3 | kaushik  |  23 | Kota  |  2000.00 |
|  4 | Chaitali |  25 | Mumbai  |  6500.00 |
|  5 | Hardik  |  27 | Bhopal  |  8500.00 |
|  6 | Komal  |  22 | Pune  |  4500.00 |
|  7 | Muffy  |  24 | Indore  | 10000.00 |
+----+----------+-----+-----------+----------+
```

If you want to modify all ADDRESS and SALARY column values in CUSTOMERS table, you do not need to use WHERE clause and UPDATE query would be as follows:

```
UPDATE CUSTOMERS SET ADDRESS = 'Pune', SALARY = 1000.00;
```

Now, CUSTOMERS table would have the following records:

```
+----+----------+-----+---------+---------+
| ID | NAME  | AGE | ADDRESS | SALARY  |
+----+----------+-----+---------+---------+
|  1 | Ramesh  |  32 | Pune  | 1000.00 |
|  2 | Khilan  |  25 | Pune  | 1000.00 |
|  3 | kaushik  |  23 | Pune  | 1000.00 |
|  4 | Chaitali |  25 | Pune  | 1000.00 |
|  5 | Hardik  |  27 | Pune  | 1000.00 |
|  6 | Komal  |  22 | Pune  | 1000.00 |
|  7 | Muffy  |  24 | Pune  | 1000.00 |
+----+----------+-----+---------+---------+
```

3.4. Data deletion

SQL DELETE Query

The SQL DELETE Query is used to delete the existing records from a table.

You can use WHERE clause with DELETE query to delete selected rows, otherwise all the records would be deleted.

The basic syntax of DELETE query with WHERE clause is as follows:

```
DELETE FROM table_name WHERE [condition];
```

You can combine N number of conditions using AND or OR operators.

Example: Consider the CUSTOMERS table having the following records:

```
+----+----------+-----+-----------+----------+
| ID | NAME  | AGE | ADDRESS  | SALARY  |
+----+----------+-----+-----------+----------+
|  1 | Ramesh  |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan  |  25 | Delhi  |  1500.00 |
|  3 | kaushik  |  23 | Kota  |  2000.00 |
|  4 | Chaitali |  25 | Mumbai  |  6500.00 |
|  5 | Hardik  |  27 | Bhopal  |  8500.00 |
|  6 | Komal  |  22 | MP  |  4500.00 |
|  7 | Muffy  |  24 | Indore  | 10000.00 |
+----+----------+-----+-----------+----------+
```

Following is an example, which would DELETE a customer, whose ID is 6:

```
DELETE FROM CUSTOMERS WHERE ID = 6;
```

Now, CUSTOMERS table would have the following records:

```
+----+----------+-----+-----------+----------+
| ID | NAME  | AGE | ADDRESS  | SALARY  |
+----+----------+-----+-----------+----------+
|  1 | Ramesh  |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan  |  25 | Delhi  |  1500.00 |
|  3 | kaushik  |  23 | Kota  |  2000.00 |
|  4 | Chaitali |  25 | Mumbai  |  6500.00 |
|  5 | Hardik  |  27 | Bhopal  |  8500.00 |
|  7 | Muffy  |  24 | Indore  | 10000.00 |
+----+----------+-----+-----------+----------+
```

If you want to DELETE all the records from CUSTOMERS table, you do not need to use WHERE clause and DELETE query would be as follows:

DELETE FROM CUSTOMERS;

Now, CUSTOMERS table would not have any record

Self check 3

Part I: choose the best answer

1. Which SQL statement is used to insert data into a database?

A) MODIFY

B) ALTER

C) UPDATE

D) Insert

2. Which SQL statement is used to update data in a database?

A) MODIFY

B) ALTER

C) UPDATE

D) CHANGE

3. Which SQL command is used to remove all records from a table without removing the table itself?

A) DELETE

B) TRUNCATE

C) REMOVE

D) DROP

Operation sheet 3.1 Data Insertion

Operation Title: Data Insertion

Purpose: to insert data in the table

Equipment and tools: Computer, DBMS software

Steps to insert a data

Step 1: Insert the following data into student table

STUDENT Table

StudentID	Name	Sex	BirthDate	Section	DeptName	EmailAddress
R101	Yohanis	Male	20/02/80	Room1	Computer Science	john@gmail.com
R102	Kalkidan	Female	12/06/78	Room1	Computer Science	kal@yahoo.com
R103	Gari	Male	17/01/70	Room2	Electrical	gari@yahoo.com
R104	Firew	Male	10/09/73	Room1	Computer Science	Fire@gmail.com
R105	Tedi	Male	19/01/82	Room2	Electrical	tedi@gmail.com
R106	Solomon	Male	15/03/85	Room2	Computer Science	Sol@gmail.com

```
insert into student values('R101', 'Yohanis', 'Male', 20/02/80,'Room1', 'Computer Science',
        'john@gmail.com')
insert into student values('R102', 'Kalkidan', 'Female',12/06/78,'Room1', 'Computer Science',
        'kal@yahoo.com')
insert into student values ('R103', 'Gari', 'Male', 17/01/70,'Room2','Electrical', 'gari@yahoo.com')

insert into student values ('R104', 'Firew','Male', 10/09/73,'Room1','Computer Science',
        'fire@gmail.com')

insert into student values ('R105', 'Tedi', 'Female',19/01/82,'Room2', 'Electrical',
        'tedi@gmail.com')
```

```
insert into student values('R106','Solomon',  'Male',  2/05/88,'Room2','Computer Science',
        'sol@gmail.com')

insert into student values('r107','sara','female',1/01/89,'room3','accounting','sara@gmail.com')
```

Step 2: Insert the following data into Course table

COURSE Table

Course_Code	Course_Title	Credit
ICT001	PHP	80
ICT002	Java	140
Elec003	Electrical	200

```
insert into  course values('ICT001',    'PHP',  80)
insert into  course values('ICT002',    'Java',  140)
insert into  course values( 'Elec003',  'Electrical',      200)
```

Step 3: Step 1:Insert the following data into Grade_report table

GRADE_REPORT Table

SID	C_Code	Grade
R101	ICT001	B
R101	ICT002	C
R102	ICT001	A
R103	Elec003	C
R104	ICT001	B
R104	ICT002	A
R105	Elec003	B
R106	ICT001	A
R106	ICT002	B

```
insert into grade_report values( 'R101',  'ICT001',     'B')

insert into grade_report values('R101',   'ICT002',     'C')

insert into grade_report values ( 'R102', 'ICT001',     'A')

insert into grade_report values('R103',   'Elec003',    'C')

insert into grade_report values('R104',   'ICT001',     'B')
```

```
insert into grade_report values('R104',  'ICT002',     'A')

insert into grade_report values('R105',  'Elec003',    'B')
insert into grade_report values( 'R106', 'ICT001',     'A')

insert into grade_report values('R106',  'ICT002',     'B')
```

Step 4: Create a SQL command to change the student birthdate for students that belongs to Electrical department

```
update student set bithdate='4/6/1984' where deptname=’Electrical’
```

Lap Test

Instruction: Given necessary tools and materials you are required to perform the following tasks accordingly

Task 1: Create a database for called AB_Supermarket

Task 2: Create and insert the following data to the Customers and customer_order table

Customer Table

Id	FirstName	LastName	City	Country	Phone
1	Marimawit	Anderaw	Addis ababa	Ethiopia	+2519453234
2	Ana	deriba	México	Addis ababa	+2519653234
3	Anatoli	Mesgana	Adama	Ethiopia	+251978567434
4	Thomas	Haregawi	Holeta	Ethiopia	+251947894334
5	Christina	Berihun	Ferensay	Addis Ababa	0921-12 34 65
6	Hanna	Markos	Cherkos	Addis ababa	0721-086460

Customer_order Table

Id	OrderDate	CustomerId	TotalAmount	OrderNumber
1	2012-07-04 00:00:00.000	2	440.00	542378
2	2012-07-05 00:00:00.000	1	1863.40	542379
3	2012-07-08 00:00:00.000	4	1813.00	542380
4	2012-07-08 00:00:00.000	3	670.80	542381

Unit Four: Data query language

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

- Selection of data from a single table
- Selection of data from multiple tables
- Retrieval of data selectively
- Working with functions
- Working with sub-queries

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:

- Explain Data query language
- Recognize the basic syntax and structure of SQL queries.
- Write SELECT statements to retrieve data from one or more tables.
- Apply filtering conditions using WHERE clause in queries.
- Sort query results using ORDER BY clause.
- Solve business problems using SQL queries.
- Apply aggregate functions (e.g., COUNT, SUM, AVG, MAX, MIN) to analyze data.
- Group data using GROUP BY clause.
- Write subqueries to retrieve data for more complex queries.
- Understand the relationship between main queries and subqueries.

4.1. Overview of SQL data query language

In many ways, queries are the heart of the SQL language. The SELECT statement, which is used to express SQL queries, is the most powerful and complex of the SQL statements.

Despite the many options afforded by the SELECT statement, it's possible to start simply and then work up to more complex queries.

The SELECT statement retrieves data from a database and returns it to you in the form of query results.

For simple queries, the English language request and the SQL SELECT statement are very similar. When the requests become more complex, more features of the SELECT statement must be used to specify the query precisely. The full form of the SELECT statement consists of six clauses. The SELECT and FROM clauses of the statement are required. The remaining four clauses are optional. You include them in a SELECT statement only when you want to use the functions they provide.

The SELECT clause lists the data items to be retrieved by the SELECT statement. The items may be columns from the database, or columns to be calculated by SQL as it performs the query.

The SELECT clause that begins each SELECT statement specifies the data items to be retrieved by the query. The items are usually specified by a select list, a list of select items separated by commas. Each select item in the list generates a single column of query results, in left-to-right order. A select item can be: • a column name, identifying a column from the table(s) named in the FROM clause.

When a column name appears as a select item, SQL simply takes the value of that column from each row of the database table and places it in the corresponding row of query results.

• a constant, specifying that the same constant value is to appear in every row of the query results.

• a SQL expression, indicating that SQL must calculate the value to be placed into the query results, in the style specified by the expression. The result is stored in a result table, called the result-set.

Each type of select item is described later in this unit.

The FROM clause lists the tables that contain the data to be retrieved by the query. Queries tcan draw their data from a single or more tables.

The FROM clause consists of the keyword FROM, followed by a list of table specifications separated by commas. Each table specification identifies a table containing data to be retrieved by the query. These tables

are called the source tables of the query (and of the SELECT statement) because they are the source of all of the data in the query results.

The WHERE clause tells SQL to include only certain rows of data in the query results. A search condition is used to specify the desired rows.

The GROUP BYclause specifies a summary query. Instead of producing one row of query results for each row of data in the database, a summary query groups together similar rows and then produces one summary row of query results for each group.

The HAVING clause tells SQL to include only certain groups produced by the GROUP BY clause in the query results. Like the WHERE clause, it uses a search condition to specify the desired groups.

The ORDER BYclause sorts the query results based on the data in one or more
columns. If it is omitted, the query results are not sorted.

Query Results

The result of a SQL query is always a table of data, just like the tables in the database. If you type a SELECT statement using interactive SQL, the DBMS displays the query results in tabular form on your computer screen. If a program sends a query to the DBMS using programmatic SQL, the table of query results is returned to the program. In either case, the query results always have the same tabular, row/column format as the actual tables in the database

4.2. Selection of data from a single table

SQL SELECT Syntax

```
SELECT column_name(s)
FROM table_name
```

and

```
SELECT * FROM table_name
```

Note: SQL is not case sensitive. SELECT is the same as select.

An SQL SELECT Example, The "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

Now we want to select the content of the columns named "LastName" and "FirstName" from the table above.

We use the following SELECT statement:

```
SELECT LastName, FirstName FROM Persons
```

The result-set will look like this:

LastName	FirstName
Hansen	Ola
Svendson	Tove
Pettersen	Kari

SELECT * Example

Now we want to select all the columns from the "Persons" table. We use the following SELECT statement:

```
SELECT * FROM Persons
```

Tip: The asterisk (*) is a quick way of selecting all columns! The result-set will look like this:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

- **Navigation in a Result-set**

Most database software systems allow navigation in the result-set with programming functions, like: Move-To-First-Record, Get-Record-Content, Move-To-Next-Record, etc.

Programming functions like these are not a part of this tutorial.

4.3. Retrieval of data selectively

- **SQL SELECT DISTINCT Statement**

In a table, some of the columns may contain duplicate values. This is not a problem, however, sometimes you will want to list only the different (distinct) values in a table.

The DISTINCT keyword can be used to return only distinct (different) values.

SQL SELECT DISTINCT Syntax

```
SELECT DISTINCT column_name(s)  FROM table_name
```

SELECT DISTINCT Example: The "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

Now we want to select only the distinct values from the column named "City" from the table above.

We use the following SELECT statement:

```
SELECT DISTINCT City FROM Persons
```

The result-set will look like this:

City
Sandnes
Stavanger

- **SQL WHERE Clause**
  - ✓ The WHERE clause is used to filter records.

The WHERE clause is used to extract only those records that fulfill a specified criterion.

SQL WHERE Syntax

```
SELECT column_name(s)
FROM table_name

WHERE column_name operator value
```

WHERE Clause Example

The "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

Now we want to select only the persons living in the city "Sandnes" from the table above.
We use the following SELECT statement:

```
SELECT * FROM Persons WHERE City='Sandnes'
```

- **Quotes around Text Fields**

SQL uses single quotes around text values (most database systems will also accept double quotes).
Although, numeric values should not be enclosed in quotes. For
text values:

```
This is correct:
SELECT * FROM Persons WHERE FirstName='Tove'
This is wrong:
SELECT * FROM Persons WHERE FirstName=Tove
```

For numeric values:

```
This is correct:
SELECT * FROM Persons WHERE Year=1965
This is wrong:
SELECT * FROM Persons WHERE Year='1965'
```

Operators Allowed in the WHERE Clause

With the WHERE clause, the following operators can be used:

Operator	Description
=	Equal
<>	Not equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
BETWEEN	Between an inclusive range
LIKE	Search for a pattern
IN	If you know the exact value you want to return for at least one of the columns

Note: In some versions of SQL the <> operator may be written as !=

- **SQL AND & OR Operators**

The AND & OR operators are used to filter records based on more than one condition.

The AND operator displays a record if both the first and the second condition is true.

The OR operator displays a record if either the first condition or the second condition is true.

AND Operator Example

The "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

Now we want to select only the persons with the first name equal to "Tove" AND the last name equal to "Svendson":

We use the following SELECT statement:

```
SELECT * FROM Persons
WHERE FirstName='Tove'
AND LastName='Svendson'
```

The result-set will look like this:

P_Id	LastName	FirstName	Address	City
2	Svendson	Tove	Borgvn 23	Sandnes

OR Operator Example

Now we want to select only the persons with the first name equal to "Tove" OR the first name equal to "Ola":

We use the following SELECT statement:

```
SELECT * FROM Persons
WHERE FirstName='Tove'
OR FirstName='Ola'
```

The result-set will look like this:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes

Combining AND & OR

You can also combine AND and OR (use parenthesis to form complex expressions).

Now we want to select only the persons with the last name equal to "Svendson" AND the first name equal to "Tove" OR to "Ola":

We use the following SELECT statement:

```
SELECT * FROM Persons WHERE
LastName='Svendson'
AND (FirstName='Tove' OR FirstName='Ola')
```

The result-set will look like this:

P_Id	LastName	FirstName	Address	City
2	Svendson	Tove	Borgvn 23	Sandnes

- **SQL ORDER BY Keyword**

The ORDER BY keyword is used to sort the result-set.

The ORDER BY keyword is used to sort the result-set by a specified column. The ORDER BY keyword sort the records in ascending order by default.

If you want to sort the records in a descending order, you can use the DESC keyword.

SQL ORDER BY Syntax

```
SELECT column_name(s)
FROM table_name

ORDER BY column_name(s) ASC|DESC
```

ORDER BY Example

The "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger
4	Nilsen	Tom	Vingvn 23	Stavanger

Now we want to select all the persons from the table above, however, we want to sort the persons by their last name.

We use the following SELECT statement:

```
SELECT * FROM Persons
ORDER BY LastName
```

The result-set will look like this:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
4	Nilsen	Tom	Vingvn 23	Stavanger
3	Pettersen	Kari	Storgt 20	Stavanger
2	Svendson	Tove	Borgvn 23	Sandnes

ORDER BY DESC Example

Now we want to select all the persons from the table above, however, we want to sort the persons descending by their last name.

We use the following SELECT statement:

```
SELECT * FROM Persons
ORDER BY LastName DESC
```

The result-set will look like this:

P_Id	LastName	FirstName	Address	City
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger
4	Nilsen	Tom	Vingvn 23	Stavanger
1	Hansen	Ola	Timoteivn 10	Sandnes

- **SQL UPDATE Statement**

The UPDATE statement is used to update existing records in a table.

SQL UPDATE Syntax

```
UPDATE table_name
SET column1=value, column2=value2,...
WHERE some_column=some_value
```

Note: Notice the WHERE clause in the UPDATE syntax. The WHERE clause specifies which record or records that should be updated. If you omit the WHERE clause, all records will be updated!

SQL UPDATE Example

The "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger
4	Nilsen	Johan	Bakken 2	Stavanger
5	Tjessem	Jakob		

Now we want to update the person "Tjessem, Jakob" in the "Persons" table.

We use the following SQL statement:

```
UPDATE Persons
SET Address='Nissestien 67', City='Sandnes'
WHERE LastName='Tjessem' AND FirstName='Jakob'
```

The "Persons" table will now look like this:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger
4	Nilsen	Johan	Bakken 2	Stavanger
5	Tjessem	Jakob	Nissestien 67	Sandnes

- **SQL UPDATE Warning**

Be careful when updating records. If we had omitted the WHERE clause in the example above, like this:

```
UPDATE Persons
SET Address='Nissestien 67', City='Sandnes'
```

The "Persons" table would have looked like this:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Nissestien 67	Sandnes
2	Svendson	Tove	Nissestien 67	Sandnes
3	Pettersen	Kari	Nissestien 67	Sandnes
4	Nilsen	Johan	Nissestien 67	Sandnes
5	Tjessem	Jakob	Nissestien 67	Sandnes

- **SQL DELETE Statement**

The DELETE statement is used to delete rows/ records in a table.

SQL DELETE Syntax

```
DELETE FROM table_name
WHERE some_column=some_value
```

Note: Notice the WHERE clause in the DELETE syntax. The WHERE clause specifies which record or records that should be deleted. If you omit the WHERE clause, all records will be deleted!

SQL DELETE Example

The "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger
4	Nilsen	Johan	Bakken 2	Stavanger
5	Tjessem	Jakob	Nissestien 67	Sandnes

Now we want to delete the person "Tjessem, Jakob" in the "Persons" table. We use the following SQL statement:

```
DELETE FROM Persons
WHERE LastName='Tjessem' AND FirstName='Jakob'
```

The "Persons" table will now look like this:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger
4	Nilsen	Johan	Bakken 2	Stavanger

- Delete All Rows

It is possible to delete all rows in a table without deleting the table. This means that the table structure, attributes, and indexes will be intact:

```
DELETE  FROM  table_name
or
DELETE * FROM table_name
```

Note: Be very careful when deleting records. You cannot undo this statement!

1. **Character and String Comparison Operators:**

- = (Equal to): Compares if two character or string values are identical.
- != or <> (Not equal to): Checks if two character or string values are not identical.
- LIKE: Compares a character or string value with a pattern, often using wildcard characters (% and _).

Example:

```
SELECT * FROM customers WHERE last_name = 'Smith';
SELECT * FROM products WHERE product_name LIKE 'App%';
```

- **Date and Time Comparison Operators:**

➢ **= (Equal to): Compares if two date or time values are equal.**

➢ != or <> (Not equal to): Compares if two date or time values are not equal.

➢ < (Less than): Checks if one date or time value is earlier than another.

➢ > (Greater than): Checks if one date or time value is later than another.

➢ <= (Less than or equal to): Checks if one date or time value is earlier than or equal to another.

- >= (Greater than or equal to): Checks if one date or time value is later than or equal to another.

Example:

```
SELECT * FROM orders WHERE order_date >= '2023-01-01';
```

When using these comparison operators in your SQL queries, be sure to match the data types appropriately. For example, when comparing dates, make sure the date format matches the one used in your database. Also, consider the collation (sorting and comparison rules) for character and string data when using comparison operators for text-based data.

To control the order of evaluation and ensure correct precedence, you can use parentheses to group conditions.

Example:

SELECT * FROM employees WHERE (age >= 30 OR experience_years >= 5) AND department = 'Engineering';

In this query, the conditions inside the parentheses are evaluated first, and then the AND condition is applied to the results.

Using parentheses is especially important when combining AND and OR operators in the same query to make your intentions explicit and avoid unexpected behavior.

Understanding the correct precedence and using parentheses when necessary is crucial to ensure that your SQL queries produce the desired results and correctly evaluate complex conditions.

Checking for a range of values

You can use comparison operators to filter rows based on a range of values. For example, to retrieve products with prices between $50 and $100:

SELECT * FROM products WHERE price >= 50 AND price <= 100;

This query retrieves rows from the "products" table where the "price" is greater than or equal to 50 and less than or equal to 100.

- **Selecting values from a list**

You can use the **IN** operator to filter rows based on a list of values. For example, to retrieve orders from customers in New York or California:

SELECT * FROM orders WHERE customer_state IN ('New York', 'California');

This query retrieves rows from the "orders" table where the "customer_state" is either 'New York' or 'California'.

- **Checking for Values that Match a Pattern**

You can use the LIKE operator with wildcard characters to match values that fit a specific pattern. For example, to retrieve products with names starting with 'Laptop':

SELECT * FROM products WHERE product_name LIKE 'Laptop%';

This query retrieves rows from the "products" table where the "product_name" starts with 'Laptop'. The '%' wildcard matches any sequence of characters, so it matches 'Laptop,' 'Laptop Pro,' 'Laptop 2023,' and so on.

You can also use the NOT LIKE operator to exclude rows that match a particular pattern. For example, to retrieve products with names that don't start with 'Accessory':

SELECT * FROM products WHERE product_name NOT LIKE 'Accessory%';

This query retrieves rows where the "product_name" does not start with 'Accessory'.

These SQL techniques are commonly used to filter and retrieve specific data from a database based on various criteria, making it easier to work with the data that meets your specific requirements.

- **Taking action to execute null values from a query result**

In SQL, you can take action to handle and filter out null values from a query result using the **IS NULL** and **IS NOT NULL** operators. Here's how you can use them:

Filtering Rows with Null Values (IS NULL):

To retrieve rows that contain null values in a specific column, you can use the **IS NULL** operator. For example, to retrieve all products with null values in the "description" column:

```
SELECT * FROM products WHERE description IS NULL;
```

This query will return rows where the "description" column is null.

- **Filtering Rows without Null Values (IS NOT NULL):**

To retrieve rows that do not contain null values in a specific column, you can use the **IS NOT NULL** operator. For example, to retrieve all products with a non-null value in the "price" column:

SELECT * FROM products WHERE price IS NOT NULL;

This query will return rows where the "price" column is not null.

- **Handling Null Values in the Result (COALESCE):**

If you want to replace null values in the query result with a specific default value, you can use the **COALESCE** function. For example, if you want to replace null values in the "description" column with "No description available":

SELECT product_name, COALESCE(description, 'No description available') AS description

FROM products;

This query uses the **COALESCE** function to replace null values with the specified default value in the result set.

Handling null values is important to ensure the accuracy and completeness of your query results. The methods mentioned above allow you to filter and manage null values effectively in your SQL queries.

4.4. Working with functions

- **Using arithmetical operators with the correct precedence**

➢ **Arithmetic Operators**

You can use arithmetic operators in Multidimensional Expressions (MDX) for any arithmetic computations, including addition, subtraction, multiplication, and division.

MDX supports the arithmetic operators listed in the following table.

Operator	Description
+ (Add)	Adds two numbers.
/ (Divide)	Divides one number by another number.
* (Multiply)	Multiplies two numbers.
- (Subtract)	Subtracts two numbers.
^ (Power)	Raises one number by another number.

Note: MDX does not include a function to obtain the square root of a number. To obtain the square root of a number, raise it to the power of 0.5 using the ^ operatior.

- **Order of Precedence**

The following rules determine the order of precedence for arithmetic operators in an MDX expression:

- When there is more than one arithmetic operator in an expression, MDX performs multiplication and division first, followed by subtraction and addition.
- When all arithmetic operators in an expression have the same level of precedence, the order of execution is left to right.

- Expressions within parentheses take precedence over all other operations.

In SQL, you can perform arithmetic operations using operators such as **+**, **-**, *****, and **/**. It's important to understand operator precedence when combining multiple operators in an expression.

Example: Suppose you have a table called "Products" with columns "Price" and "Quantity." You want to calculate the total cost for each product, considering the unit price and quantity. The SQL statement would be:

```
SELECT ProductName, (Price * Quantity) AS TotalCost  FROM Products;
```

In this example, the * operator is used to multiply the "Price" and "Quantity" columns to obtain the "TotalCost."

- **Using String Functions and Operators:**

SQL provides various string functions and operators for working with text data. For instance, you can use the CONCAT function to concatenate strings.

Example:

Suppose you have a table called "Employees" with columns "FirstName" and "LastName." You want to create a single string representing the full name. The SQL statement would be:

```
SELECT CONCAT(FirstName, ' ', LastName) AS FullName
FROM Employees;
```

In this example, the CONCAT function is used to combine the "FirstName" and "LastName" columns with a space in between.

- **Using Mathematical Functions:**

SQL offers mathematical functions for performing operations like rounding, absolute value, square root, etc.

Example:

Suppose you have a table called "Orders" with a "TotalAmount" column. You want to calculate the square root of the total amount. The SQL statement would be:

SELECT OrderID, SQRT(TotalAmount) AS SquareRootAmount FROM Orders;

In this example, the SQRT function calculates the square root of the "TotalAmount."

- **Using Date Functions:**

SQL provides various date functions for working with date and time data.

Example:

Suppose you have a table called "Events" with a "EventDate" column, and you want to find the events that occurred within the last 30 days. The SQL statement would be:

SELECT EventName, EventDate FROM Events WHERE EventDate >= DATEADD(day, -30, GETDATE());

In this example, the DATEADD function subtracts 30 days from the current date (GETDATE()), and the WHERE clause filters events that occurred after that date.

- **Using SQL Aggregate Functions:**

SQL aggregate functions are used to perform calculations on sets of values.

AVG () - The AVG() function returns the average value of a numeric column.

AVG() Syntax

```
SELECT AVG(column_name) FROM table_name WHERE condition;
```

Example

```
SELECT AVG(Price)
FROM Products;
```

COUNT() – The COUNT() function returns the number of rows that matches a specified criterion.

COUNT() Syntax

SELECT COUNT(*column_name*) FROM *table_name* WHERE *condition*;

Example

```
SELECT COUNT(*)
FROM Products;
```

MAX() - Returns the largest value

MIN() - Returns the smallest value

SUM() - The `SUM()` function returns the total sum of a numeric column.

SUM() Syntax

```
SELECT SUM(column_name) FROM table_name
WHERE condition;
```

Example:

Suppose you have a table called "Orders" with an "OrderAmount" column, and you want to find the total sales for a specific period. The SQL statement would be:

```
SELECT SUM(OrderAmount) AS TotalSales
FROM Orders
WHERE OrderDate BETWEEN '2023-01-01' AND '2023-12-31';
```

In this example, the SUM function calculates the total sales for orders placed between January 1, 2023, and December 31, 2023.

These examples demonstrate the use of arithmetical, string, mathematical, date, and aggregate functions in SQL queries to obtain the desired query output, depending on the data and calculations you need.

- **SQL statements that use aggregation and filtering**

Using Clause to Aggregate Data by Multiple Columns:

When you want to aggregate data based on multiple columns, you can use the GROUP BY clause. This clause allows you to group rows by one or more columns and apply aggregate fun **SQL**

- **GROUP BY Statement**

The GROUP BY statement is used in conjunction with the aggregate functions to group the result-set by one or more columns.

SQL GROUP BY Syntax

```
SELECT column_name, aggregate_function(column_name)
FROM table_name
WHERE column_name operator value
GROUP BY column_name
```

SQL GROUP BY Example1

We have the following "Orders" table:

O_Id	OrderDate	OrderPrice	Customer
1	2008/11/12	1000	Hansen
2	2008/10/23	1600	Nilsen
3	2008/09/02	700	Hansen
4	2008/09/03	300	Hansen
5	2008/08/30	2000	Jensen
6	2008/10/04	100	Nilsen

Now we want to find the total sum (total order) of each customer.

We will have to use the GROUP BY statement to group the customers. We use the following SQL statement:

```
SELECT Customer,SUM(OrderPrice) FROM Orders
GROUP BY Customer
```

The result-set will look like this:

Customer	SUM(OrderPrice)
Hansen	2000
Nilsen	1700
Jensen	2000

Now we want to find if the customers "Hansen" or "Jensen" have a total order of more than 1500.

- **Having clause**

We add an ordinary WHERE clause to the SQL statement:

```
SELECT Customer,SUM(OrderPrice) FROM Orders
GROUP BY Customer
HAVING SUM(OrderPrice)>1500
```

The result-set will look like this

Customer	SUM(OrderPrice)
Hansen	2000
Jensen	2000

- **GROUP BY More Than One Column**

We can also use the GROUP BY statement on more than one column, like this:

Suppose you have a table called "Sales" with columns "Product," "Region," and "Revenue." To aggregate data by both "Product" and "Region" and calculate the total revenue for each combination, you can use the following SQL statement:

```
SELECT Product, Region, SUM(Revenue) AS TotalRevenue
FROM Sales
GROUP BY Product, Region;
```

In this example, the data is grouped by both "Product" and "Region," and the SUM function is applied to calculate the total revenue for each group.

- **Sorting Aggregate Data in Query Output:**

You can sort the aggregate data in the query output using the ORDER BY clause. This allows you to specify the order in which the results should be displayed.

Use ORDER BY if you want to order rows according to a value returned by an aggregate function like SUM(). The ORDER BY operator is followed by the aggregate function (in our example, SUM()). DESC is placed after this function to specify a descending sort order. Thus, the highest aggregate values are displayed first, then progressively lower values are displayed. To sort in ascending order, you can specify ASC or simply omit either keyword, as ascending is the default sort order.

Example1:

Continuing with the "Sales" table, if you want to see the total revenue for each product and region combination, sorted in descending order of revenue, you can use the following SQL statement:

```
SELECT Product, Region, SUM(Revenue) AS TotalRevenue
FROM Sales
GROUP BY Product, Region
ORDER BY TotalRevenue DESC;
```

In this example, the data is first aggregated, and then the results are sorted in descending order of the "TotalRevenue" column.

- **Filtering Aggregate Data Using the HAVING Clause:**

The HAVING clause is used to filter the results of aggregate functions. It allows you to specify conditions that the aggregated data must meet.

SQL HAVING Syntax

- SELECT column_name, aggregate_function(column_name)
- FROM table_name
- WHERE column_name operator value
- GROUP BY column_name
- HAVING aggregate_function(column_name) operator value

SQL HAVING Example1

We have the following "Orders" table:

O_Id	OrderDate	OrderPrice	Customer
1	2008/11/12	1000	Hansen
2	2008/10/23	1600	Nilsen
3	2008/09/02	700	Hansen
4	2008/09/03	300	Hansen
5	2008/08/30	2000	Jensen
6	2008/10/04	100	Nilsen

Now we want to find if any of the customers have a total order of less than 2000. We use the following SQL statement:

```
SELECT Customer,SUM(OrderPrice) FROM Orders
GROUP BY Customer
HAVING SUM(OrderPrice)<2000
```

The result-set will look like this:

Customer	SUM(OrderPrice)
Nilsen	1700

Example2:

Suppose you want to find product-region combinations with a total revenue greater than $10,000. You can use the following SQL statement:

```
SELECT Product, Region, SUM(Revenue) AS TotalRevenue FROM Sales
GROUP BY Product, Region
HAVING TotalRevenue > 10000;
```

In this example, the HAVING clause filters the results to include only those combinations where the "TotalRevenue" is greater than $10,000.

These SQL statements demonstrate how to aggregate data, sort the results, and filter aggregated data using the GROUP BY, ORDER BY, and HAVING clauses, respectively. These clauses are essential for summarizing and manipulating data in SQL queries.

4.5. Working with subqueries

4.5.1 Single row subquery

- **SQL Sub Queries**

A Subquery or Inner query or Nested query is a query within another SQL query and embedded within the WHERE clause.

A subquery is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved.

Subqueries can be used with the SELECT, INSERT, UPDATE, and DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN etc.

There are a few rules that subqueries must follow:

- Subqueries must be enclosed within parentheses.
- A subquery can have only one column in the SELECT clause, unless multiple columns are in the main query
- for the subquery to compare its selected columns.
- An ORDER BY cannot be used in a subquery, although the main query can use an ORDER BY. The GROUP
- BY can be used to perform the same function as the ORDER BY in a subquery.
- Subqueries that return more than one row can only be used with multiple value operators, such as the IN operator.
- The SELECT list cannot include any references to values that evaluate to a BLOB, ARRAY, CLOB or NCLOB.
- A subquery cannot be immediately enclosed in a set function.
- The BETWEEN operator cannot be used with a subquery; however, the BETWEEN operator can be used within the subquery. Subqueries with the SELECT Statement:
- Subqueries are most frequently used with the SELECT statement. The basic syntax is as follows:

SELECT column_name [, column_name]

FROM table1 [, table2]

WHERE column_name OPERATOR

Self-check 4

Part I: Choose the correct answer

1. Which SQL statement is used to retrieve data from a database?

A) SEARCH

B) EXTRACT

C) SELECT

D) RETRIEVE

2. What is the purpose of the SQL WHERE clause in a SELECT statement?

A) To specify the columns to be retrieved

B) To filter the rows based on a specified condition

C) To order the result set in ascending order

D) To join multiple tables

3. In SQL, what is the purpose of the GROUP BY clause?

A) To filter rows based on a condition

B) To order the result set

C) To group rows that have the same values in specified columns

D) To join tables

4. What does the SQL acronym "JOIN" refer to?

A) Combine

B) Concatenate

C) Merge

D) List

Operation sheet 4.1 Inserting data into a database

Operation Title: Inserting data to a database

Purpose: To populate database with a data

Equipment and tools: Computer and DBMS software

Steps to insert data to the table

Step 1. Write a query to display the student information for all students.

```
select*from student
```

Step 2. Write a query that displays course_code and course_title from the Course table.

```
select Course_code, Course_title from course
```

Step 3. Write a query to display the student name for all students those who score grades 'B'.

```
SELECT student.Name, grade_report.Grade FROM course INNER JOIN
grade_report ON course.Course_Code = grade_report.Course_Code INNER JOIN
student ON grade_report.StudentID = student.StudentID
WHERE (grade_report.Grade = 'B')
```

Step 4. Write a query to display the student ID and name for all male students those who score grades 'A' and who learn in Java course.

```
SELECT student.StudentID, student.Name FROM course INNER JOIN
grade_report ON course.Course_Code = grade_report.Course_Code INNER JOIN student ON
grade_report.StudentID = student.StudentID WHERE (student.Sex = 'male') AND
(course.Course_Code = 'ICT002') AND (grade_report.Grade = 'A')
```

Step 5. Write a query to display the student for all students those who score grades ‘A’ and who learn in computer science department, then sort by Department name descending and name ascending order.

```
SELECT student.StudentID, student.Name, student.DeptName, grade_report.Grade FROM course INNER JOIN grade_report ON course.Course_Code = grade_report.Course_Code INNER JOIN student ON grade_report.StudentID = student.StudentID WHERE (student.DeptName = 'Computer Science') AND (grade_report.Grade = 'A')
ORDER BY student.DeptName DESC, student.Name
```

Step 7. Write a query to display the student for all students those who score grades between ‘A’ and ‘D’ and who learn in PhP course.

```
select * from student where StudentID in(select  distinct StudentID from grade_report where Course_Code='ICT001' and grade='A'or Course_Code='ICT001' and grade='b' or Course_Code='ICT001' and grade='c'or Course_Code='ICT001' and grade='d' )
```

Step 8. Write a query to display the student ID and name for all students who study in a computer science department with any student whose name contains a ‘T’.

```
SELECT studentid ,NAME FROM student WHERE DeptName='Computer Science' or Name LIKE'%T%'
```

Step 9. Write a query to display the student information who didn’t take a course.

```
select * from student where StudentID not in(select studentID from grade_report )
```

Step 10. Write a query to display the student information who didn’t score c.

```
select * from student where StudentID in(select studentID from grade_report where grade<>'c' )
```

Step 11. Write a query that displays only the unique grade letters.

```
select distinct grade from  grade_report
```

Step 12. Write a query that displays studentid and no of course for students who takes more than one course.

```
select studentID,count(*)total from grade_report group by studentid having count(*)>1
```

Step 13. Write a query that displays students information who takes more than one course.

```
select * from student where StudentID in(select studentID from grade_report group by studentid having count(*)>1 )
```

Lap Test

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

Task 1: Display customers first name and last name whose country is Addis ababa

Task 2: Multiply total amount column by 1.1 to get the tax included amount and display as tax_included_amount

Task 3: Delete customer_order whose order number ORD123

Task 4: Update the City of a customer in to Addis ababa whose last name is Haregawi

Task 5: Sort the customer_order in descending order by its order date

Task 6: Group orders by CustomerId and calculate the total amount for each customer

Task 7: Retrieve the maximum total amount from customer_Order using a subquery

Reference

Books

Database System Concepts

Database Systems Complete Book

Fundamental_of_Database_Systems

SQL - The Complete Reference

a_taylor_sql_for_dummies_2003

Database-Design-2nd-Edition-1660153697

SQL_All-in-One_For_Dummies.pdf

URL

https://www.tutorialspoint.com/

https://www.w3schools.com/sql/default.asp#gsc.tab=0&gsc.q=date%20functions%20sql

http://www-db.deis.unibo.it/courses/TW/DOCS/w3schools/sql/sql_select.asp.html

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