UNIT - I
INTRODUCTION

Introduction to DBMS

Database is a collection of data and Management System is a set of programs to store and retrieve those data. Based on this we can define DBMS like this: DBMS is a collection of inter-related data and set of programs to store & access those data in an easy and effective manner.

What is the need of DBMS?

Database systems are basically developed for large amount of data. When dealing with huge amount of data, there are two things that require optimization: Storage of data and retrieval of data.

Storage:

According to the principles of database systems, the data is stored in such a way that it acquires lot less space as the redundant data (duplicate data) has been removed before storage.

Let's take a example to understand this:

In a banking system, suppose a customer is having two accounts, one is saving account and another is salary account. Let's say bank stores saving account data at one place (these places are called tables we will learn them later) and salary account data at another place, in that case if the customer information such as customer name, address etc. are stored at both places then this is just a wastage of storage (redundancy/ duplication of data), to organize the data in a better way the information should be stored at one place and both the accounts should be linked to that information somehow. The same thing we achieve in DBMS.

Fast Retrieval of data:

Along with storing the data in an optimized and systematic manner, it is also important that we retrieve the data quickly when needed. Database systems ensure that the data is retrieved as quickly as possible.

Purpose of Database Systems

The main purpose of database systems is to manage the data. Consider a university that keeps the data of students, teachers, courses, books etc. To manage this data we need to store this data somewhere where we can add new data, delete unused data, update outdated data, retrieve data, to perform these operations on data we need a Database management system that allows us to store the data in such a way so that all these operations can be performed on the data efficiently.

Database Applications – DBMS

Applications where we use Database Management Systems are:

1. Telecom:

There is a database to keeps track of the information regarding calls made, network usage, customer details etc. Without the database systems it is hard to maintain that huge amount of data that keeps updating every millisecond.

2. Industry:

Where it is a manufacturing unit, warehouse or distribution centre, each one needs a database to keep the records of ins and outs. For example distribution centre should keep a track of the product units that supplied into the centre as well as the products that got delivered out from the distribution centre on each day; this is where DBMS comes into picture.

3. Banking System:

For storing customer info, tracking day to day credit and debit transactions, generating bank statements etc. All this work has been done with the help of Database management systems.

4. Sales:

To store customer information, production information and invoice details.

5. Airlines:

To travel though airlines, we make early reservations; this reservation information along with flight schedule is stored in database.

6. Education sector:

Database systems are frequently used in schools and colleges to store and retrieve the data regarding student details, staff details, course details, exam details, payroll data, attendance details, fees details etc. There is a hell lot amount of inter-related data that needs to be stored and retrieved in an efficient manner.

7. Online shopping:

Online shopping websites such as Amazon, Flip kart etc. These sites store the product information, customer addresses and preferences, credit details and provide the relevant list of products based on query. All this involves a Database management system.

Advantages of DBMS over file system

what is a file processing system and how Database management systems are better than file processing systems will be discussed here.

Drawbacks of File system

1. Data redundancy:

Data redundancy refers to the duplication of data, lets say we are managing the data of a college where a student is enrolled for two courses, the same student details in such case will be stored twice, which will take more storage than needed. Data redundancy often leads to higher storage costs and poor access time.

2. Data inconsistency:

Data redundancy leads to data inconsistency, lets take the same example that we have taken above, a student is enrolled for two courses and we have student address stored twice, now lets say student requests to change his address, if the address is changed at one place and not on all the records then this can lead to data inconsistency.

3. Data Isolation:

Because data are scattered in various files, and files may be in different formats, writing new application programs to retrieve the appropriate data is difficult.

4. Dependency on application programs:

Changing files would lead to change in application programs.

5. Atomicity issues:

Atomicity of a transaction refers to “All or nothing”, which means either all the operations in a transaction executes or none.

The architecture of DBMS depends on the computer system on which it runs. For example, in client-server DBMS architecture, the database systems at server machine can run several requests made by client machine

Types of DBMS Architecture

There are three types of DBMS architecture:

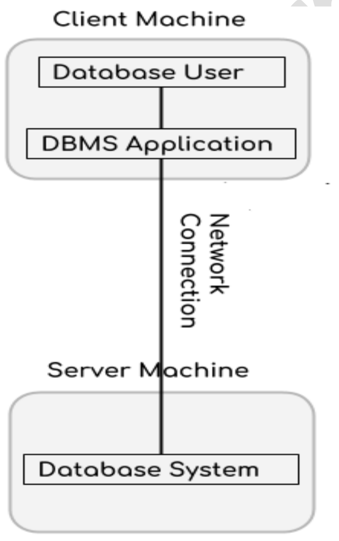
1. Single tier architecture
2. Two tier architecture
3. Three tier architecture

1. Single tier architecture:

In this type of architecture, the database is readily available on the client machine, any request made by client doesn’t require a network connection to perform the action on the database.

For example, lets say you want to fetch the records of employee from the database and the database is available on your computer system, so the request to fetch employee details will be done by your computer and the records will be fetched from the database by your computer as well. This type of system is generally referred as local database system.

2. Two tier architecture:

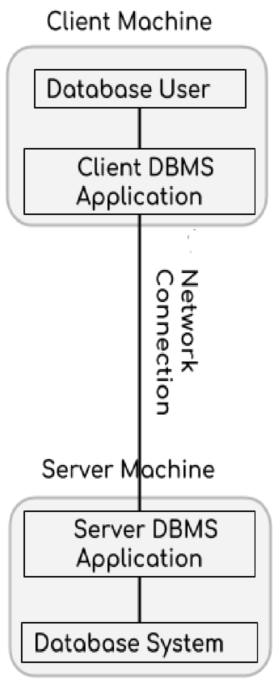


Two-Tier architecture

In two-tier architecture, the Database system is present at the server machine and the DBMS application is present at the client machine, these two machines are connected with each other through a reliable network as shown in the above diagram.

Whenever client machine makes a request to access the database present at server using a query language like sql, the server perform the request on the database and returns the result back to the client. The application connection interface such as JDBC, ODBC are used for the interaction between server and client.

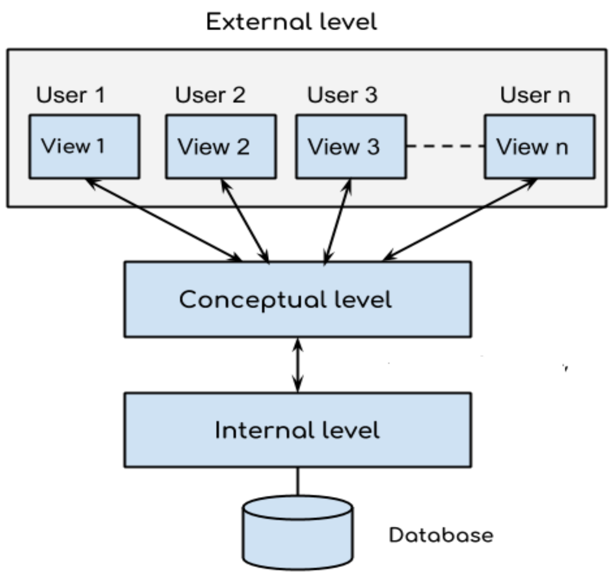
3. Three tier architecture



Three-Tier architecture

In three-tier architecture, another layer is present between the client machine and server machine. In this architecture, the client application doesn't communicate directly with the database systems present at the server machine, rather the client application communicates with server application and the server application internally communicates with the database system present at the server

DBMS Three Level Architecture Diagram:



This architecture has three levels:

1. External level
2. Conceptual level

3. Internal level

1. External level

It is also called view level. The reason this level is called "view" is because several users can view their desired data from this level which is internally fetched from database with the help of conceptual and internal level mapping.

The user doesn't need to know the database schema details such as data structure, table definition etc. user is only concerned about data which is what returned back to the view level after it has been fetched from database (present at the internal level).
External level is the "top level" of the Three Level DBMS Architecture.

2. Conceptual level:

It is also called logical level. The whole design of the database such as relationship among data, schema of data etc. are described in this level.

Database constraints and security are also implemented in this level of architecture. This level is maintained by DBA (database administrator).

3. Internal level:

This level is also known as physical level. This level describes how the data is actually stored in the storage devices. This level is also responsible for allocating space to the data. This is the lowest level of the architecture.

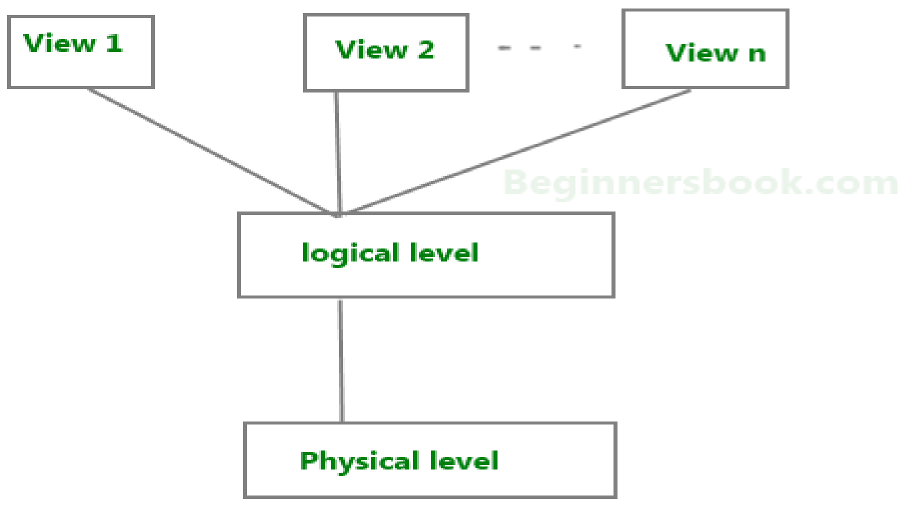
View of Data in DBMS

Abstraction is one of the main features of database systems. Hiding irrelevant details from user and providing abstract view of data to users, helps in easy and efficient user-database interaction. In the previous tutorial, we discussed the three level of DBMS architecture, The top level of that architecture is "view level". The view level provides the "view of data" to the users and hides the irrelevant details such as data relationship, database schema, constraints, security etc from the user.

1. Data abstraction
2. Instance and schema

Data Abstraction in DBMS

Database systems are made-up of complex data structures. To ease the user interaction with database, the developers hide internal irrelevant details from users. This process of hiding irrelevant details from user is called data abstraction.



Three Levels of data abstraction

We have three levels of abstraction:

Physical level:

This is the lowest level of data abstraction. It describes how data is actually stored in database. You can get the complex data structure details at this level.

Logical level:

This is the middle level of 3-level data abstraction architecture. It describes what data is stored in database.

View level:

Highest level of data abstraction. This level describes the user interaction with database system.

Example:

Let's say we are storing customer information in a customer table. At physical level these records can be described as blocks of storage (bytes, gigabytes, terabytes etc.) in memory. These details are often hidden from the programmers.

At the logical level these records can be described as fields and attributes along with their data types, their relationship among each other can be logically implemented. The programmers generally work at this level because they are aware of such things about database systems.

At view level, user just interact with system with the help of GUI and enter the details at the screen, they are not aware of how the data is stored and what data is stored; such details are hidden from them.

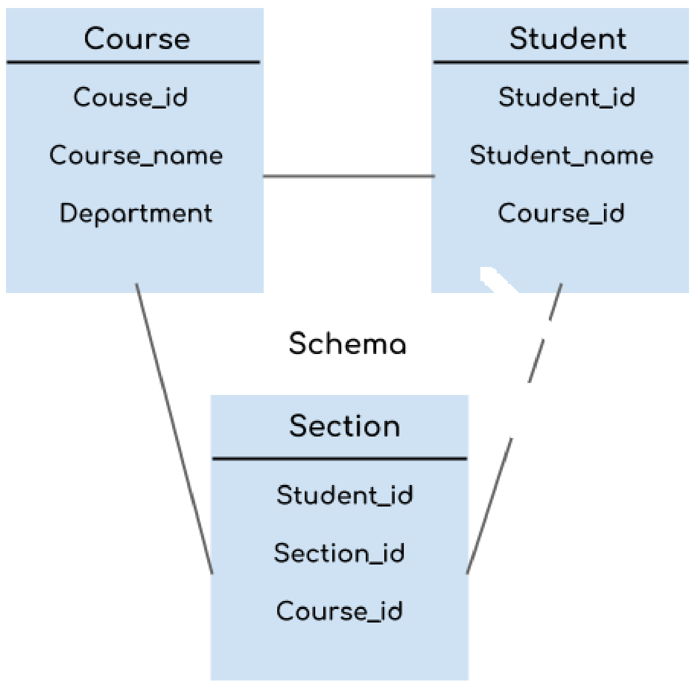
DBMS Schema

Definition of schema:

Design of a database is called the schema. Schema is of three types: Physical schema, logical schema and view schema.

For example:

In the following diagram, we have a schema that shows the relationship between three tables: Course, Student and Section. The diagram only shows the design of the database, it doesn't show the data present in those tables. Schema is only a structural view(design) of a database as shown in the diagram below.



The design of a database at physical level is called physical schema, how the data stored in blocks of storage is described at this level.

Design of database at logical level is called logical schema, programmers and database administrators work at this level, at this level data can be described as certain types of data records gets stored in data structures, however the internal details such as implementation of data structure is hidden at this level (available at physical level).

Design of database at view level is called view schema. This generally describes end user interaction with database systems.

To learn more about these schemas, refer 3 level data abstraction architecture.

DBMS Instance

Definition of instance:

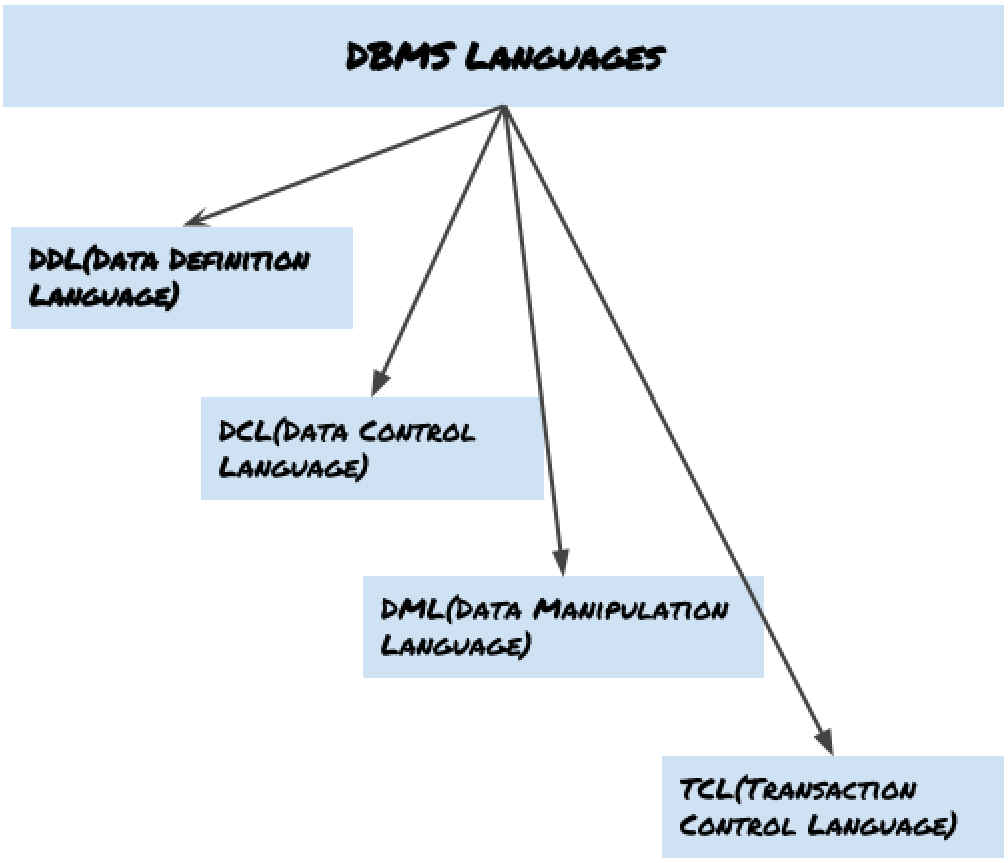
The data stored in database at a particular moment of time is called instance of database. Database schema defines the variable declarations in tables that belong to a particular database; the value of these variables at a moment of time is called the instance of that database.

For example, lets say we have a single table student in the database, today the table has 100 records, so today the instance of the database has 100 records. Lets say we are going to add another 100 records in this table by tomorrow so the instance of database tomorrow will have 200 records in table. In short, at a particular moment the data stored in database is called the instance that changes over time when we add or delete data from the database.

DBMS languages

Database languages are used to read, update and store data in a database. There are several such languages that can be used for this purpose; one of them is SQL (Structured Query Language).

Types of DBMS languages:


DBMS LANGUAGES
DDL(DATA DEFINITION LANGUAGE)
DCL(DATA CONTROL LANGUAGE)
DML(DATA MANIPULATION LANGUAGE)
TCL(TRANSACTION CONTROL LANGUAGE)

Data Definition Language (DDL)

DDL is used for specifying the database schema. It is used for creating tables, schema, indexes, constraints etc. in database. Let’s see the operations that we can perform on database using DDL:

1. To create the database instance – CREATE
2. To alter the structure of database – ALTER
3. To drop database instances – DROP
4. To delete tables in a database instance – TRUNCATE
5. To rename database instances – RENAME
6. To drop objects from database such as tables – DROP
7. To Comment – Comment

All of these commands either defines or update the database schema that's why they come under Data Definition language.

Data Manipulation Language (DML)

DML is used for accessing and manipulating data in a database. The following operations on data base come under DML:

1. To read records from table(s) – SELECT
2. To insert record(s) into the table(s) – INSERT
3. Update the data in table(s) – UPDATE
4. Delete all the records from the table – DELETE

Data Control language (DCL)

DCL is used for granting and revoking user access on a database –

1. To grant access to user – GRANT
2. To revoke access from user – REVOKE

In practical data definition language, data manipulation language and data control languages are not separate language, rather they are the parts of a single database language such as SQL.

Transaction Control Language (TCL)

The changes in the database that we made using DML commands are either performed or roll backed using TCL.

1. To persist the changes made by DML commands in database – COMMIT
2. To rollback the changes made to the database – ROLLBACK

UNIT – II
INTRODUCTION TO THE RELATIONAL MODEL AND SQL

Data Base Architecture:

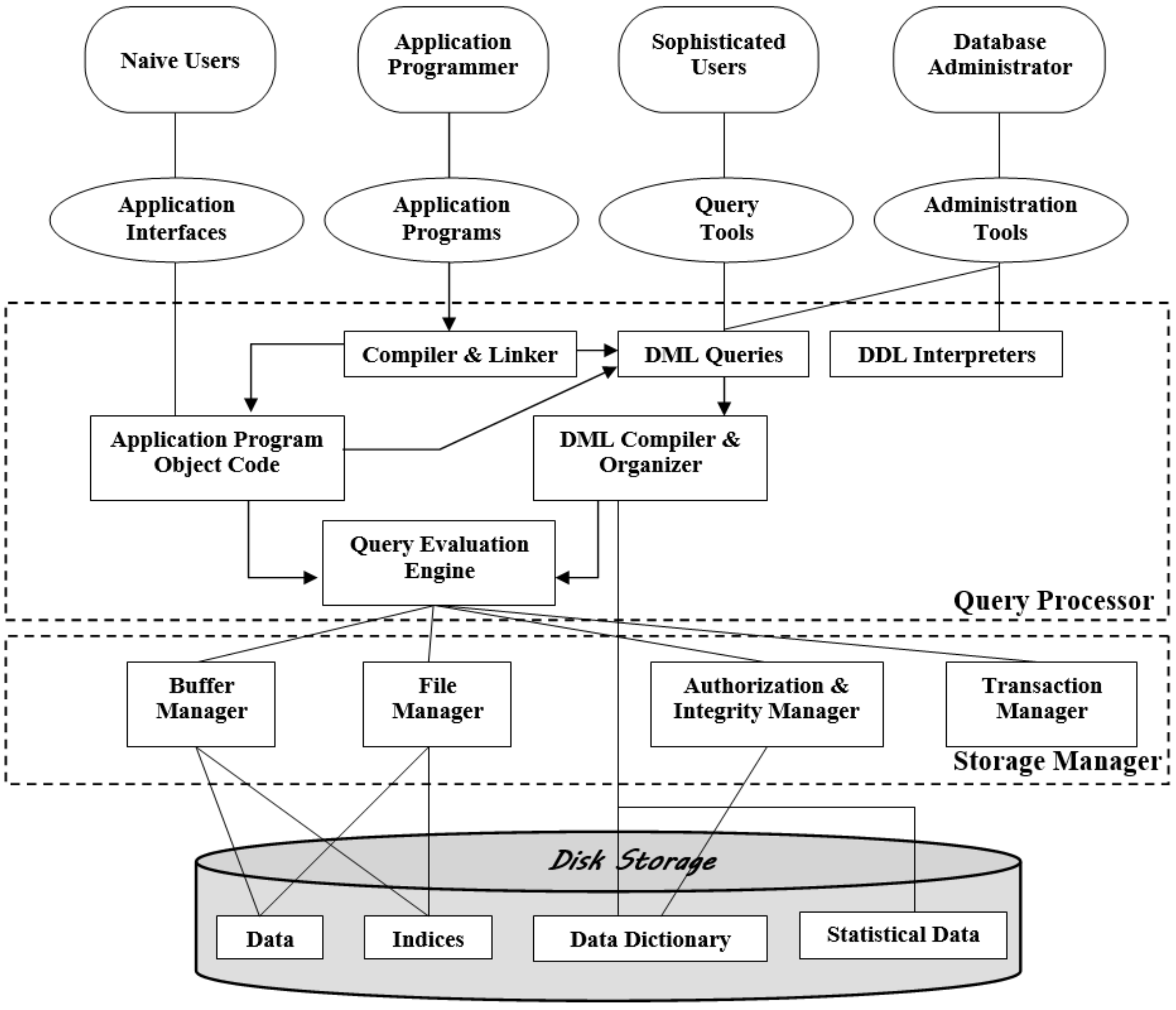


Figure: System Architecture

A database system is partitioned into modules that deal with each of the responsibilities of the overall system. The functional components of a database system can be broadly divided into the storage manager and the query processor components.

The storage manager is important because databases typically require a large amount of storage space. Corporate databases range in size from hundreds of gigabytes to, for the largest databases, terabytes of data. A gigabyte is approximately 1000 megabytes (actually 1024) (1 billion bytes), and a terabyte is 1 million megabytes (1 trillion bytes). Since the main memory of computers cannot store this much information, the information is stored on disks. Data are moved between disk storage and main memory as needed. Since the movement of data to and from disk is slow relative to the speed of the central processing unit, it is imperative that the database system structure the data so as to minimize the need to move data between disk and main memory.

The query processor is important because it helps the database system to simplify and facilitate access to data. The query processor allows database users to obtain good performance while being able to work at the view level and not be burdened with understanding the physical-level details of the implementation of the system. It is the job of the database system to translate updates and queries written in a nonprocedural language, at the logical level, into an efficient sequence of operations at the physical level.

Storage Manager

The storage manager is the component of a database system that provides the interface between the low-level data stored in the database and the application programs and queries submitted to the system. The storage manager is responsible for the interaction with the file manager. The raw data are stored on the disk using the file system provided by the operating system. The storage manager translates the various DML statements into low-level file-system commands. Thus, the storage manager is responsible for storing, retrieving, and updating data in the database.

The storage manager components include:

1. Authorization and integrity manager, which tests for the satisfaction of integrity constraints and checks the authority of users to access data
2. Transaction manager, which ensures that the database remains in a consistent (correct) state despite system failures, and that concurrent transaction executions proceed without conflicting.
3. File manager, which manages the allocation of space on disk storage and the data structures used to represent information stored on disk.
4. Buffer manager, which is responsible for fetching data from disk storage into main memory, and deciding what data to cache in main memory. The buffer manager is a critical part of the database system, since it enables the database to handle data sizes that are much larger than the size of main memory.

The storage manager implements several data structures as part of the physical system implementation:

5. Data files, which store the database itself. Data dictionary, which stores metadata about the structure of the database, in particular the schema of the data base
6. Indices, which can provide fast access to data items. Like the index in this textbook, a database index provides pointers to those data items that hold a particular value. For example, we could use an index to find the instructor record with a particular ID, or all instructor records with a particular name. Hashing is an alternative to indexing that is faster in some but not all cases.

The Query Processor:
The query processor components include:

DDL interpreter, which interprets DDL statements and records the definitions in the data dictionary

DML compiler, which translates DML statements in a query language into an evaluation plan consisting of low-level instructions that the query evaluation engine understands

A query can usually be translated into any of a number of alternative evaluation plans that all give the same result. The DML compiler also performs query optimization; that is, it picks the lowest cost evaluation plan from among the alternatives.

Query evaluation engine, which executes low-level instructions generated by the DML compiler.

Database design

Database design mainly involves the design of the database schema. The design of a complete database application environment that meets the needs of the enterprise being modeled requires attention to a broader set of issues. In this text, we focus initially on the writing of database queries and the design of database schemas.

Design Process

A high-level data model provides the database designer with a conceptual frame work in which to specify the data requirements of the database users, and how the database will be structured to fulfill these requirements. The initial phase of database design, then, is to characterize fully the data needs of the prospective database users. The database designer needs to interact extensively with domain experts and users to carry out this task. The outcome of this phase is a specification of user requirements.

Next, the designer chooses a data model, and by applying the concepts of the chosen data model, translates these requirements into a conceptual schema of the database. The schema developed at this conceptual-design phase provides a detailed overview of the enterprise. The designer reviews the schema to confirm that all data requirements are indeed satisfied and are not in conflict with one another. The designer can also examine the design to remove any redundant features. The focus at this point is on describing the data and their relationships, rather than on specifying physical storage details.

In terms of the relational model, the conceptual-design process involves decisions on what attributes we want to capture in the database and how to group these attributes to form the various tables. The “what” part is basically a business decision, and we shall not discuss it further in this text. The “how” part is mainly a computer-science problem. There are principally two ways to tackle the problem. The first one is to use the entity-relationship model; the other is to employ a set of algorithms (collectively known as normalization) that takes as input the set of all attributes and generates a set of tables.

A fully developed conceptual schema indicates the functional requirements of the enterprise. In a specification of functional requirements, users describe the kinds of operations (or transactions) that will be performed on the data. Example operations include modifying or updating data, searching for and retrieving specific data, and deleting data. At this stage of conceptual design, the designer can review the schema to ensure it meets functional requirements.

The process of moving from an abstract data model to the implementation of the database proceeds in two final design phases. In the logical-design phase, the designer maps the high-level conceptual schema onto the implementation data model of the database system that will be used. The designer uses the resulting system-specific database schema in the subsequent physical-design phase, in which the physical features of the database are specified. These features include the form of file organization and the internal storage structures.

Database Design for a University Organization:

To illustrate the design process, let us examine how a database for a university could be designed. The initial specification of user requirements may be based on interviews with the database users, and on the designer’s own analysis of the organization. The description that arises from this design phase serves as the basis for specifying the conceptual structure of the database. Here are the major characteristics of the university.

a) The university is organized into departments. Each department is identified by a unique name (dept_name), is located in a particular building, and has a budget.

b) Each department has a list of courses it offers. Each course has associated with it a course id, title, dept_name, and credits, and may also have have associated prerequisites.

c) Instructors are identified by their unique ID. Each instructor has name, associated department (dept_name), and salary.

d) Students are identified by their unique ID. Each student has a name, an associated major department (dept_name), and tot_cred (total credit hours the student earned thus far).

e) The university maintains a list of classrooms, specifying the name of the building, room_number, and room capacity.

f) The university maintains a list of all classes (sections) taught. Each section is identified by a course_id, sec_id, year, and semester, and has associated with it a semester, year, building, room_number, and time_slot _d (the time slot when the class meets).

g) The department has a list of teaching assignments specifying, for each instructor, the sections the instructor is teaching.

h) The university has a list of all student course registrations, specifying, for each student, the courses and the associated sections that the student has taken (registered for).

A real university database would be much more complex than the preceding design. However we use this simplified model to help you understand conceptual

Transaction Management

A transaction is a collection of operations that performs a single logical function in a database application. Each transaction is a unit of both atomicity and consistency. Thus, we require that transactions do not violate any database consistency constraints. That is, if the database was consistent when a transaction started, the database must be consistent when the transaction successfully terminates. However, during the execution of a transaction, it may be necessary temporarily to allow inconsistency, since either the debit of A or the credit of B must be done before the other. This temporary inconsistency, although necessary, may lead to difficulty if a failure occurs.

It is the programmer’s responsibility to define properly the various transactions, so that each preserves the consistency of the database. For example, the transaction to transfer funds from the account of department A to the account of department B could be defined to be composed of two separate programs: one that debits account A, and another that credits account B. The execution of these two programs one after the other will indeed preserve consistency. However, each program by itself does not transform the database from a consistent state to a new consistent state. Thus, those programs are not transactions.

Ensuring the atomicity and durability properties is the responsibility of the database system itself—specifically, of the recovery manager. In the absence of failures, all transactions complete successfully, and atomicity is achieved easily. However, because of various types of failure, a transaction may not always complete its execution successfully. If we are to ensure the atomicity property, a failed transaction must have no effect on the state of the database. Thus, the database must be restored to the state in which it was before the transaction in question started executing. The database system must therefore perform failure recovery, that is, detect system failures and restore the database to the state that existed prior to the occurrence of the failure.

Finally, when several transactions update the database concurrently, the consistency of data may no longer be preserved, even though each individual transaction is correct. It is the responsibility of the concurrency-control manager to control the interaction among the concurrent transactions, to ensure the consistency of the database. The transaction manager consists of the concurrency-control manager and the recovery manager.

The concept of a transaction has been applied broadly in database systems and applications. While the initial use of transactions was in financial applications, the concept is now used in real-time applications in telecommunication, as well as in the management of long-duration activities such as product design or administrative workflows.

Data Mining and Information Retrieval

The term data mining refers loosely to the process of semi automatically analyzing large databases to find useful patterns. Like knowledge discovery in artificial intelligence (also called machine learning) or statistical analysis, data mining attempts to discover rules and patterns from data. However, data mining differs from machine learning and statistics in that it deals with large volumes of data, stored primarily on disk. That is, data mining deals with "knowledge discovery in databases."

Some types of knowledge discovered from a database can be represented by a set of rules. The following is an example of a rule, stated informally: "Young womenwith annual incomes greater than $50,000 are the most likely people to buy small sports cars." Of course such rules are not universally true, but rather have degrees of "support" and "confidence." Other types of knowledge are represented by equations relating different variables to each other, or by other mechanisms for predicting outcomes when the values of some variables are known.

There are a variety of possible types of patterns that may be useful, and different techniques are used to find different types of patterns. In Chapter 20 we study a few examples of patterns and see how they may be automatically derived from a database.

Usually there is a manual component to data mining, consisting of preprocessing data to a form acceptable to the algorithms, and post processing of discovered patterns to find novel ones that could be useful. There may also be more than one type of pattern that can be discovered from a given database, and manual interaction may be needed to pick useful types of patterns. For this reason, data mining is really a semiautomatic process in real life. However, in our description we concentrate on the automatic aspect of mining.

Businesses have begun to exploit the burgeoning data online to make better decisions about their activities, such as what items to stock and how best to target customers to increase sales. Many of their queries are rather complicated, however, and certain types of information cannot be extracted even by using SQL.

Several techniques and tools are available to help with decision support. Several tools for data analysis allow analysts to view data in different ways. Other analysis tools precompute summaries of very large amounts of data, in order to give fast responses to queries. The SQL standard contains additional constructs to support data analysis.

Large companies have diverse sources of data that they need to use for making business decisions. To execute queries efficiently on such diverse data, companies have built data warehouses. Data warehouses gather data from multiple sources under a unified schema, at a single site. Thus, they provide the user a single uniform interface to data.

Textual data, too, has grown explosively. Textual data is unstructured, unlike the rigidly structured data in relational databases. Querying of unstructured textual data is referred to as information retrieval. Information retrieval systems have much in common with database systems—in particular, the storage and retrieval of data on secondary storage. However, the emphasis in the field of information systems is different from that in database systems, concentrating on issues such as querying based on keywords; the relevance of documents to the query; and the analysis, classification, and indexing of documents.

Database Users and Administrators

A primary goal of a database system is to retrieve information from and store new information into the database. People who work with a database can be categorized as database users or database administrators.

Database Users and User Interfaces:

There are four different types of database-system users, differentiated by the way they expect to interact with the system. Different types of user interfaces have been designed for the different types of users.

Naive users are unsophisticated users who interact with the system by invoking one of the application programs that have been written previously.

For example:

A clerk in the university who needs to add a new instructor to department A invokes a program called new hire. This program asks the clerk for the name of the new instructor, her new ID, the name of the department (that is, A), and the salary.

The typical user interface for naive users is a forms interface, where the user can fill in appropriate fields of the form. Naive users may also simply read reports generated from the database.

As another example, consider a student, who during class registration period, wishes to register for a class by using a Web interface. Such a user connects to a Web application program that runs at a Web server. The application first verifies the identity of the user, and allows her to access a form where she enters the desired information. The form information is sent back to the Web application at the server, which then determines if there is room in the class (by retrieving information from the database) and if so adds the student information to the class roster in the database.

Application programmers are computer professionals who write application programs. Application programmers can choose from many tools to develop user interfaces. Rapid application development (RAD) tools are tools that enable an application programmer to construct forms and reports with minimal programming effort.

Sophisticated users interact with the system without writing programs. Instead, they form their requests either using a database query language or by using tools such as data analysis software. Analysts who submit queries to explore data in the database fall in this category.

Specialized users are sophisticated users who write specialized database applications that do not fit into the traditional data-processing framework. Among these applications are computer-aided design systems, knowledgebase and expert systems, systems that store data with complex data types (for example, graphics data and audio data), and environment-modeling systems.

Database Administrator

One of the main reasons for using DBMSs is to have central control of both the data and the programs that access those data. A person who has such central control over the system is called a database administrator (DBA). The functions of a DBA include:

Schema definition:

The DBA creates the original database schema by executing a set of data definition statements in the DDL.

Storage structure and access - method definition. Schema and physical-organization modification. The DBA carries out changes to the schema and physical organization to reflect the changing needs of the organization, or to alter the physical organization to improve performance.

Granting of authorization for data access. By granting different types of authorization, the database administrator can regulate which parts of the database various users can access. The authorization information is kept in a special system structure that the database system consults whenever someone attempts to access the data in the system.

Routine maintenance:

Examples of the database administrator’s routine maintenance activities are:

Periodically backing up the database, either onto tapes or onto remote servers, to prevent loss of data in case of disasters such as flooding

Ensuring that enough free disk space is available for normal operations, and upgrading disk space as required

Monitoring jobs running on the database and ensuring that performance is not degraded by very expensive tasks submitted by some users.

UNIT – III
INTRODUCTION TO THE RELATIONAL MODEL AND SQL

Operators in PL/SQL

An operator is a symbol that tells the compiler to perform the operation on one or more operands specified along with the symbol. The variables on which the operation has to be performed is called operand and what operation is to be done is indicated by the operator symbol.

In PL/SQL, operators can be classified broadly into following categories:

a. Arithmetic operators
b. Relational operators
c. Comparision operators
d. Logical operators

Arithmetic Operators

Arithmetic operators are used to perform different mathematical operations on operands. Following are the arithmetic operators available in PL/SQL:

NOTE: For providing examples for every operator, let's consider two variables, a and b with values 6 and 3 respectively.

Operator	Use	Example
+	Adds the two operands	a+b will give 9
-	Performs subtraction, where the second operand is subtracted from the first operand. This operator can return negative value too.	a-b will give 3
/	Performs Division operation.	a/b will give 2
*	Performs Multiplication	a*b will give 18
**	Performs Exponentiation operation which means the first operand raised to the power the second operand	a**b will give 216

Relational Operators

Relational operators are used to compare two values and return the result in the form of boolean value(either TRUE or FALSE). They are mostly used in conditions where some sort of comparison is required. Following are the relational operators available in PL/SQL:

NOTE: For providing examples for every operator, let's consider two variables, a and b with values 7 and 14 respectively.

Operator	Use	Example
=	To check if the values of the two operands given is equal or not. If it is equal then the condition will return true else false.	(a = b) is False
!= <> ~=	These operators are used to check if the two operands are not equal to each other or they don't have the same values. If the operands are not equal then the condition will return true else false.	a != b is True
<	To check if LHS value is smaller than RHS value. If yes, condition returns true.	a < b is True
>	To check if LHS value is greater than RHS value. If yes, condition returns true.	a > b is False(not true)
<=	To check if LHS value is smaller than or equal to RHS value.	a <= b is True
>=	To check if LHS value is greater than or equal to RHS value	a >= b is False(not true)

Logical Operators

Logical operators are used to combine multiple expressions or define an expression with two operand and return either True or False based on the operands or expressions surrounding the logical operators.

NOTE: For providing examples for every operator, let's consider two variables (or expressions), a and b with values true and false respectively.

Operator	Use	Example
AND	Returns TRUE when both LHS and RHS operand are true. Returns FALSE when LHS and RHS operand are both or either of them are false.	a AND b will return false
OR	Returns TRUE when LHS and RHS operand both or either of them are true. Returns FALSE when LHS and RHS operand are false.	a OR b will return true
NOT	Applied on a single operand. Returns TRUE when operand is false. Returns FALSE when operand is true.	NOT a will return false

In the table above, when we say operand, it can be an expression too. While writing SQL queries we tend to use these operators with the WHERE clause.

Comparison Operators

Comparison operators are used to compare one value with the other to return the result as TRUE or FALSE or NULL. Following are the different types of comparison operators:

LIKE Operator

This operator is used to match a single character or group of characters (string). There are two wildcard characters which are used for the purpose of matching.
% is used to match a string of any characters.

Where as, _ is used to match a single character

You can see here: LIKE operator example in SQL

This operator returns TRUE if string or characters are matched otherwise returns FALSE.

SELECT * from student WHERE as name LIKE 'J%';

When we execute the above SQL query, it will display student record whose name starts with J where % is used to match string of any character after first character J. We will get names like Jon, Jiya, John, James etc if they are present in the student table.

Let's take another example,

SELECT * from student WHERE as name LIKE '_ _ _ a';

When we execute the above SQL query, it will display student record whose name is of 4 letters and ends with the character A.

BETWEEN Operator

This operator is used to check whether the value is within a certain given range. It returns TRUE if the value is in the given range otherwise returns FALSE.

Let's take an example:
SELECT * from student WHERE age BETWEEN 12 AND 18;

The above SQL query, will display student records whose age lies between 12 and 18.

IN Operator

This operator is used when any value is required to be compared to a given list of values. It returns TRUE if the value is present in the given list otherwise it returns FALSE. This operator comes in handy where we have to do multiple comparisons rather than using multiple OR conditions.

Let's take an example:
SELECT * from student WHERE city IN ('Delhi','Goa','Kerela');

The above SQL query, will display the student records who belongs to city Delhi or Goa or Kerela.

IS NULL Operator

This operator returns TRUE if the operand value is NULL(empty) otherwise returns FALSE

Let's take an example:
SELECT * from student WHERE age IS NULL;

The above SQL query, will display student records whose age field in table is blank (empty)

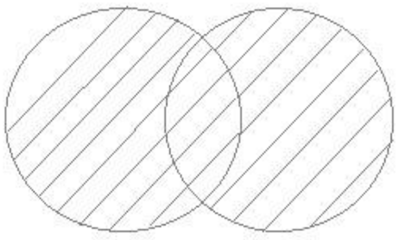
SET Operations in SQL

SQL supports few Set operations which can be performed on the table data. These are used to get meaningful results from data stored in the table, under different special conditions.

We will cover 4 different types of SET operations, along with example:
UNION
UNION ALL
INTERSECT
MINUS

UNION Operation

UNION is used to combine the results of two or more SELECT statements. However it will eliminate duplicate rows from its result set. In case of union, number of columns and data type must be same in both the tables, on which UNION operation is being applied.



Example of UNION
The First table,

ID	Name
1	abhi
2	Adam

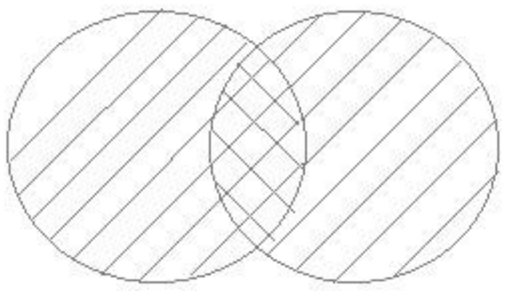
The Second table,

ID	Name
2	Adam
3	Chester

Union SQL query will be,
SELECT * FROM First
UNION
SELECT * FROM Second;
The resultset table will look like,

ID	NAME
1	Abhi
2	Adam
3	Chester

UNION ALL
This operation is similar to Union. But it also shows the duplicate rows.



Example of Union All
The First table,

ID	NAME
1	Abhi
2	Adam

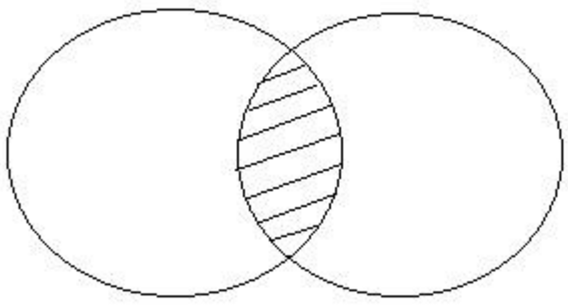
The Second table,

ID	NAME
2	Adam
3	Chester

Union All query will be like,
SELECT * FROM First
UNION ALL
SELECT * FROM Second;
The result set table will look like,

ID	NAME
1	Abhi
2	Adam
2	Adam
3	Chester

INTERSECT
Intersect operation is used to combine two SELECT statements, but it only returns the records which are common from both SELECT statements. In case of Intersect the number of columns and data type must be same.
NOTE: MySQL does not support INTERSECT operator.



Example of Intersect
The First table,

ID	NAME
1	abhi
2	Adam

The Second table,

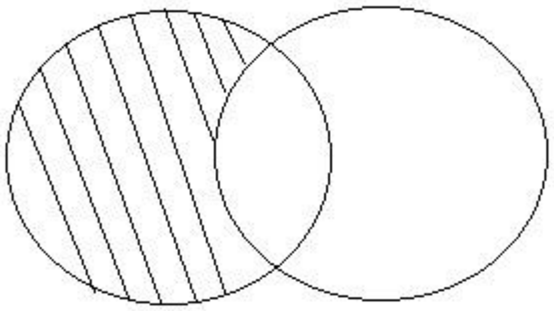
ID	NAME
2	Adam
3	Chester

Intersect query will be,
SELECT * FROM First
INTERSECT
SELECT * FROM Second;
The result set table will look like

ID	NAME
2	Adam

MINUS

The Minus operation combines results of two SELECT statements and return only those in the final result, which belongs to the first set of the result.



Example of Minus
The First table,

ID	NAME
1	Abhi
2	Adam

The Second table,

ID	NAME
2	Adam
3	Chester

Minus query will be,
SELECT * FROM First
MINUS
SELECT * FROM Second;

The result set table will look like,

ID	NAME
1	Abhi

Aggregate functions in SQL

In database management an aggregate function is a function where the values of multiple rows are grouped together as input on certain criteria to form a single value of more significant meaning.

Various Aggregate Functions:

1. Count()
2. Sum()
3. Avg()
4. Min()
5. Max()

Now let us understand each Aggregate function with a example:

```
Id    Name     Salary
-----------------------
1      A       80
2      B       40
3      C       60
4      D       70
5      E       60
6      F       Null
```

Count():
Count (*): Returns total number of records .i.e 6.
Count (salary): Return number of Non Null values over the column salary. i.e 5.
Count (Distinct Salary): Return number of distinct Non Null values over the column salary .i.e 4

Sum():
Sum (salary): Sum all Non Null values of Column salary i.e., 310
sum (Distinct salary): Sum of all distinct Non-Null values i.e., 250.

Avg ():
Avg(salary) = Sum(salary) / count(salary) = 310/5
Avg(Distinct salary) = sum(Distinct salary) / Count(Distinct Salary) = 250/4

Min ():
Min (salary): Minimum value in the salary column except NULL i.e., 40.
Max (salary): Maximum value in the salary i.e., 80.

SQL SUB QUERIES

In SQL a Subquery can be simply defined as a query within another query. In other words, we can say that a Subquery is a query that is embedded in WHERE clause of another SQL query.

Important rules for Sub queries:

1. You can place the Sub query in a number of SQL clauses: WHERE clause, HAVING clause, FROM clause.
2. Sub queries can be used with SELECT, UPDATE, INSERT, DELETE statements along with expression operator. It could be equality operator or comparison operator such as =, >, =, <= and Like operator.
3. A sub query is a query within another query. The outer query is called as main query and inner query is called as sub query.
4. The sub query generally executes first, and its output is used to complete the query condition for the main or outer query.
5. Sub query must be enclosed in parentheses.
6. Sub queries are on the right side of the comparison operator.
7. ORDER BY command cannot be used in a Sub query. GROUPBY command can be used to perform same function as ORDER BY command.
8. Use single-row operators with single row Sub queries. Use multiple-row operators with multiple-row Sub queries.

Syntax:
There is not any general syntax for Sub queries. However, Sub queries are seen to be used most frequently with SELECT statement as shown below:

```
SELECT column_name
FROM table_name
WHERE column_name expression operator
 ( SELECT COLUMN_NAME  from TABLE_NAME   WHERE ... );
```

Sample Table:
DATA BASE

NAME	ROLL_NO	LOCATION	PHONE_NUMBER
Ram	101	Chennai	9988775566
Raj	102	Coimbatore	8877665544
Sasi	103	Madurai	7766553344
Ravi	104	Salem	8989898989
Sumathi	105	Kanchipuram	8989856868

STUDENT

NAME	ROLL_NO	SECTION
Ravi	104	A
Sumathi	105	B
Raj	102	A

Sample Queries

To display NAME, LOCATION, PHONE_NUMBER of the students from DATABASE table whose section is A

Select NAME, LOCATION, PHONE_NUMBER from DATABASE
WHERE ROLL_NO IN(SELECT ROLL_NO from STUDENT where SECTION='A');

Explanation : First sub query executes " SELECT ROLL_NO from STUDENT where SECTION='A' " returns ROLL_NO from STUDENT table whose SECTION is 'A'. Then outer-query executes it and return the NAME, LOCATION, PHONE_NUMBER from the DATABASE table of the student whose ROLL_NO is returned from inner sub query.

Output:

NAME	ROLL_NO	LOCATION	PHONE_NUMBER
Ravi	104	Salem	8989898989
Raj	102	Coimbatore	8877665544

Insert Query Example:
Table1: Student1

NAME	ROLL_NO	LOCATION	PHONE_NUMBER
Ram	101	Chennai	9988773344
Raju	102	Coimbatore	9090909090
Ravi	103	Salem	8989898989

Table2: Student2

NAME	ROLL_NO	LOCATION	PHONE_NUMBER
Raj	111	Chennai	8787878787
Sai	112	Mumbai	6565656565
Sri	113	Coimbatore	7878787878

To insert Student2 into Student1 table:

INSERT INTO Student1 SELECT * FROM Student2;

Output:

NAME	ROLL_NO	LOCATION	PHONE_NUMBER
Ram	101	Chennai	9988773344
Raju	102	Coimbatore	9090909090
Ravi	103	Salem	8989898989
Raj	111	Chennai	8787878787

Sai	112	Mumbai	6565656565
Sri	113	Coimbatore	7878787878

To delete students from Student2 table whose rollno is same as that in Student1 table and having location as chennai
DELETE FROM Student2
WHERE ROLL_NO IN (SELECT ROLL_NO
FROM Student1
WHERE LOCATION = ’chennai’);
Output:
1 row delete successfully.

Display Student - 2 table:

NAME	ROLL_NO	LOCATION	PHONE_NUMBER
Sai	112	Mumbai	6565656565
Sri	113	Coimbatore	7878787878

To update name of the students to geeks in Student2 table whose location is same as Raju,Ravi in Student1 table
UPDATE Student2
SET NAME=’geeks’
WHERE LOCATION IN (SELECT LOCATION
FROM Student1
WHERE NAME IN (‘Raju’,’Ravi’));
Output:
1 row updated successfully.

Display Student2 table:

NAME	ROLL_NO	LOCATION	PHONE_NUMBER
Sai	112	Mumbai	6565656565
Geeks	113	Coimbatore	7878787878

SQL | Views
Views in SQL are kind of virtual tables. A view also has rows and columns as they are in a real table in the database. We can create a view by selecting fields from one or more tables present in the database. A View can either have all the rows of a table or specific rows based on certain condition.

Sample Tables:
Student Details

S_ID	NAME	ADDRESS
1	Harsh	Kolkata
2	Ashish	Durgapur
3	Pratik	Delhi
4	Dhanraj	Bihar
5	Ram	Rajasthan

Student Marks

ID	NAME	MARKS	AGE
1	Harsh	90	19
2	Suresh	50	20
3	Pratik	80	19
4	Dhanraj	95	21
5	Ram	85	18

CREATING VIEWS

We can create View using CREATE VIEW statement. A View can be created from a single table or multiple tables.

Syntax:

```
CREATE VIEW view_name AS
SELECT column1, column2.....
FROM table_name
WHERE condition;
```

view_name: Name for the View
table_name: Name of the table
condition: Condition to select rows

Examples:

Creating View from a single table:

In this example we will create a View named DetailsView from the table StudentDetails.

Query:

```
CREATE VIEW DetailsView AS
SELECT NAME, ADDRESS
FROM StudentDetails
WHERE S_ID < 5;
```

To see the data in the View, we can query the view in the same manner as we query a table.

```
SELECT * FROM DetailsView;
```

Output:

NAME	ADDRESS
HARSH	Kolkata
ASHISH	Durgapur
PRATIK	Delhi
DHANRAJ	Bihar

In this example, we will create a view named Student Names from the table Student Details.

Query:
CREATE VIEW Student Names AS
SELECT S_ID, NAME
FROM Student Details
ORDER BY NAME;
If we now query the view as,
SELECT * FROM Student Names;
Output:

S_ID	NAME
2	Ashish
4	Dhanraj
1	Harsh
3	Pratik
5	Ram

Creating View from multiple tables: In this example we will create a View named Marks View from two tables Student Details and Student Marks. To create a View from multiple tables we can simply include multiple tables in the SELECT statement. Query:
CREATE VIEW Marks View AS
SELECT StudentDetails.NAME, Student Details. ADDRESS, Student Marks. MARKS
FROM Student Details, Student Marks
WHERE StudentDetails.NAME = StudentMarks.NAME;
To display data of View Marks View:
SELECT * FROM Marks View;
Output:

NAME	ADDRESS	MARKS
Harsh	Kolkata	90
Pratik	Delhi	80
Dhanraj	Bihar	95
Ram	Rajasthan	85

DELETING VIEWS

We have learned about creating a View, but what if a created View is not needed any more? Obviously we will want to delete it. SQL allows us to delete an existing View. We can delete or drop a View using the DROP statement.
Syntax:

DROP VIEW view_name;
view_name: Name of the View which we want to delete.
For example, if we want to delete the View MarksView, we can do this as:
DROP VIEW MarksView;
UPDATING VIEWS
There are certain conditions needed to be satisfied to update a view. If any one of these conditions is not met, then we will not be allowed to update the view.
The SELECT statement which is used to create the view should not include GROUP BY clause or ORDER BY clause.
The SELECT statement should not have the DISTINCT keyword.

The View should have all NOT NULL values.
The view should not be created using nested queries or complex queries.
The view should be created from a single table. If the view is created using multiple tables then we will not be allowed to update the view.
We can use the CREATE OR REPLACE VIEW statement to add or remove fields from a view.
Syntax:
CREATE OR REPLACE VIEW view_name AS
SELECT column1,coulmn2,..
FROM table_name
WHERE condition;
For example, if we want to update the view MarksView and add the field AGE to this View from StudentMarks Table, we can do this as:

CREATE OR REPLACE VIEW MarksView AS
SELECT StudentDetails.NAME, Student Details. ADDRESS, Student Marks. MARKS, Student Marks. AGE
FROM Student Details, Student Marks
WHERE StudentDetails.NAME = StudentMarks.NAME;
If we fetch all the data from Marks View now as:
SELECT * FROM Marks View;

Output:

NAME	ADDRESS	MARKS	AGE
Harsh	Kolkata	90	19
Pratik	Delhi	80	19
Dhanraj	Bihar	95	21
Ram	Rajasthan	85	18

Inserting a row in a view:
We can insert a row in a View in a same way as we do in a table. We can use the INSERT INTO statement of SQL to insert a row in a View. Syntax:

INSERT INTO view_name(column1, column2 , column3,..)
VALUES (value1, value2, value3..);
view_name: Name of the View
Example:
In the below example we will insert a new row in the View Details View which we have created above in the example of “creating views from a single table”.

INSERT INTO Details View(NAME, ADDRESS)
VALUES("Suresh","Gurgaon");

If we fetch all the data from Details View now as,
SELECT * FROM Details View;

Output:

NAME	ADDRESS
Harsh	Kolkata
Ashish	Durgapur
Pratik	Delhi
Dhanraj	Bihar
Suresh	Gurgaon

Deleting a row from a View:
Deleting rows from a view is also as simple as deleting rows from a table. We can use the DELETE statement of SQL to delete rows from a view. Also deleting a row from a view first delete the row from the actual table and the change is then reflected in the view.Syntax:
DELETE FROM view_name
WHERE condition;

view_name:Name of view from where we want to delete rows
condition: Condition to select rows
Example:
In this example we will delete the last row from the view DetailsView which we just added in the above example of inserting rows.
DELETE FROM DetailsView
WHERE NAME="Suresh";
If we fetch all the data from DetailsView now as,
SELECT * FROM DetailsView;
Output:

NAME	ADDRESS
Harsh	Kolkata
Ashish	Durgapur
Pratik	Delhi
Dhanraj	Bihar

WITH CHECK OPTION
The WITH CHECK OPTION clause in SQL is a very useful clause for views. It is applicable to a updatable view. If the view is not updatable, then there is no meaning of including this clause in the CREATE VIEW statement.
The WITH CHECK OPTION clause is used to prevent the insertion of rows in the view where the condition in the WHERE clause in CREATE VIEW statement is not satisfied.
If we have used the WITH CHECK OPTION clause in the CREATE VIEW statement, and if the UPDATE or INSERT clause does not satisfy the conditions then they will return an error.
Example:
In the below example we are creating a View Sample View from Student Details Table with WITH CHECK OPTION clause.
CREATE VIEW Sample View AS
SELECT S_ID, NAME
FROM Student Details
WHERE NAME IS NOT NULL
WITH CHECK OPTION;
In this View if we now try to insert a new row with null value in the NAME column then it will

give an error because the view is created with the condition for NAME column as NOT NULL.
For example, though the View is updatable but then also the below query for this View is not valid:
INSERT INTO Sample View(S_ID)
VALUES (6);

SQL Constraints
SQL Constraints are rules used to limit the type of data that can go into a table, to maintain the accuracy and integrity of the data inside table.
Constraints can be divided into the following two types,
Column level constraints: Limits only column data.
Table level constraints: Limits whole table data.
Constraints are used to make sure that the integrity of data is maintained in the database.
Following are the most used constraints that can be applied to a table.

NOT NULL
UNIQUE
PRIMARY KEY
FOREIGN KEY
CHECK
DEFAULT

NOT NULL Constraint
NOT NULL constraint restricts a column from having a NULL value. Once NOT NULL constraint is applied to a column, you cannot pass a null value to that column. It enforces a column to contain a proper value.
One important point to note about this constraint is that it cannot be defined at table level.

Example using NOT NULL constraint
CREATE TABLE Student(s_id int NOT NULL, Name varchar(60), Age int);
The above query will declare that the s_id field of Student table will not take NULL value.

UNIQUE Constraint
UNIQUE constraint ensures that a field or column will only have unique values. A UNIQUE constraint field will not have duplicate data. This constraint can be applied at column level or table level.

Using UNIQUE constraint when creating a Table (Table Level)
Here we have a simple CREATE query to create a table, which will have a column s_id with unique values.
CREATE TABLE Student(s_id int NOT NULL UNIQUE, Name varchar(60), Age int);
The above query will declare that the s_id field of Student table will only have unique values and wont take NULL value.

Using UNIQUE constraint after Table is created (Column Level)
ALTER TABLE Student ADD UNIQUE(s_id);
The above query specifies that s_id field of Student table will only have unique value.

Primary Key Constraint

Primary key constraint uniquely identifies each record in a database. A Primary Key must contain unique value and it must not contain null value. Usually Primary Key is used to index the data inside the table.

Using PRIMARY KEY constraint at Table Level
CREATE table Student (s_id int PRIMARY KEY, Name varchar(60) NOT NULL, Age int);
The above command will creates a PRIMARY KEY on the s_id.

Using PRIMARY KEY constraint at Column Level
ALTER table Student ADD PRIMARY KEY (s_id);
The above command will creates a PRIMARY KEY on the s_id.

Foreign Key Constraint

FOREIGN KEY is used to relate two tables. FOREIGN KEY constraint is also used to restrict actions that would destroy links between tables. To understand FOREIGN KEY, let's see its use, with help of the below tables:
Customer_Detail Table

c_id	Customer_Name	Address
101	Adam	Noida
102	Alex	Delhi
103	Stuart	Rohtak

Order_Detail Table

Order_id	Order_Name	c_id
10	Order1	101
11	Order2	103
12	Order3	102

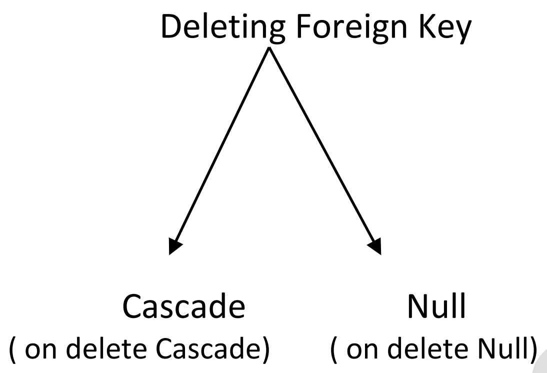
In Customer_Detail table, c_id is the primary key which is set as foreign key in Order_Detail table. The value that is entered in c_id which is set as foreign key in Order_Detail table must be present in Customer_Detail table where it is set as primary key. This prevents invalid data to be inserted into c_id column of Order_Detail table. If you try to insert any incorrect data, DBMS will return error and will not allow you to insert the data.

Using FOREIGN KEY constraint at Table Level
CREATE table Order_Detail (
Order_id int PRIMARY KEY,
Order_name varchar (60) NOT NULL,
C_id int FOREIGN KEY REFERENCES Customer Detail(c_id)
);
In this query, c_id in table Order_Detail is made as foriegn key, which is a reference

of c_id column in Customer_Detail table.

Using FOREIGN KEY constraint at Column Level
ALTER table Order_Detail ADD FOREIGN KEY (c_id) REFERENCES Customer_Detail(c_id);

Behaviour of Foriegn Key Column on Delete
There are two ways to maintin the integrity of data in Child table, when a particular record is deleted in the main table. When two tables are connected with Foriegn key, and certain data in the main table is deleted, for which a record exits in the child table, then we must have some mechanism to save the integrity of data in the child table.



On Delete Cascade : This will remove the record from child table, if that value of foriegn key is deleted from the main table.
On Delete Null : This will set all the values in that record of child table as NULL, for which the value of foriegn key is deleted from the main table.
If we don't use any of the above, then we cannot delete data from the main table for which data in child table exists. We will get an error if we try to do so.
ERROR : Record in child table exist

CHECK Constraint
CHECK constraint is used to restrict the value of a column between a range. It performs check on the values, before storing them into the database. Its like condition checking before saving data into a column.

Using CHECK constraint at Table Level
```
CREATE table Student(
 s_id int NOT NULL CHECK(s_id > 0),
 Name varchar(60) NOT NULL,
Age int
);
```
The above query will restrict the s_id value to be greater than zero.

Using CHECK constraint at Column Level
ALTER table Student ADD CHECK(s_id > 0);

Data types in PL/SQL

Data type defines the type of data being used, whether it is a number or a word (string) or a single character etc. Following data types can be used in PL/SQL depending upon the type of data required

NUMBER(p,s)
Range: p= 1 to 38 s= -84 to 127
This datatype is used to store numeric data. Here, p is precision s is scale.

Example:
Age NUMBER (2); where Age is a variable that can store 2 digits percentage NUMBER(4,2); where, percentage is a variable that can store 4 (p) digits before decimal and 2 (s) digits after decimal.

CHAR(size)
Range: 1 to 2000 bytes
a. This datatype is used to store alphabetical string of fixed length.
b. Its value is quoted in single quotes.
c. Occupies the whole declared size of memory even if the space is not utilized by the data.

Example:
rank CHAR(10); where, rank is a variable that can store upto 10 characters. If the length of data(charcaters) stored in rank is 5 then it will still occupy all the 10 spaces. 5 space in the memory will get used and the rest blank memory spaces will be wasted.

VARCHAR (size)
Range: 1 to 2000 bytes
a. This datatype is used to store alphanumeric string of variable length.
b. Its value is quoted in single quotes.
c. Occupies the whole declared size of memory even if the space is not utilized by the data.

Example:
address VARCHAR(10); where, address is a variable that can occupy maximum 10 bytes of memory space and can store alphanumeric value in it. Unused spaces are wasted.

VARCHAR2(size)
Range: 1 to 4000 bytes
a. This datatype is used to store alphanumeric string of variable length.
b. Its value is quoted in single quotes.
c. It releases the unused space in memory, hence saving the unused space.

Example:
name VARCHAR2(10); where, name is a variable that can occupy maximum 10 bytes of memory to store an alphanumeric value. The unused memory space is released.

DATE
Range: 01-Jan-4712 BC to 31-DEC-9999
a. It stores the data in date format DD-MON-YYYY
b. The value for this data type is written in single quotes.

Example:

DOB DATE; where, DOB is a variable that stores date of birth in defined format (i.e,'13-FEB-1991')

%TYPE

a. It stores value of that variable whose data type is unknown and when we want the variable to inherit the data type of the table column.
b. Also, its value is generally being retrieved from an existing table in the data base, hence it takes the data type of the column for which it is used.

Example:

Student sno %TYPE;, where Student is the name of the table created in database and sno is variable whose datatype is unknown and %TYPE is used to store its value.

BOOLEAN

a. This datatype is used in conditional statements.
b. It stores logical values.
c. It can be either TRUE or FALSE

Example:

Is Admin BOOLEAN; where, is Admin is a variable whose value can be TRUE or FALSE depending upon the condition being checked.

UNIT - IV
ENTITY RELATIONSHIP (E-R) MODELLING

Entity Relationship Diagram – ER Diagram in DBMS

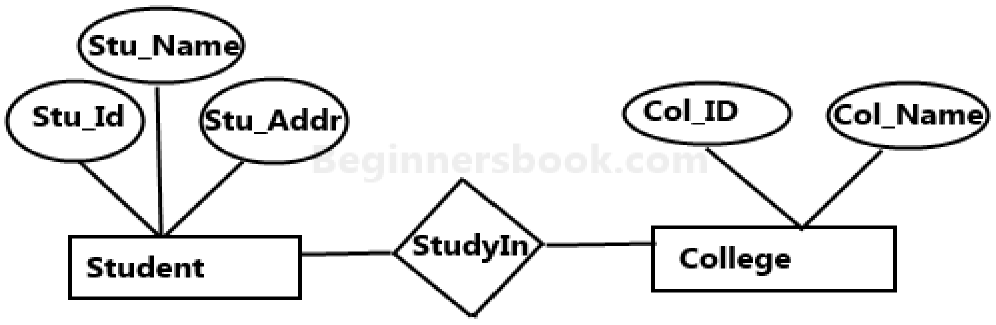
An Entity–relationship model (ER model) describes the structure of a database with the help of a diagram, which is known as Entity Relationship Diagram (ER Diagram). An ER model is a design or blueprint of a database that can later be implemented as a database. The main components of E-R model are: entity set and relationship set.

What is an Entity Relationship Diagram (ER Diagram)

An ER diagram shows the relationship among entity sets. An entity set is a group of similar entities and these entities can have attributes. In terms of DBMS, an entity is a table or attribute of a table in database, so by showing relationship among tables and their attributes, ER diagram shows the complete logical structure of a database. Lets have a look at a simple ER diagram to understand this concept.

A simple ER Diagram:

In the following diagram we have two entities Student and College and their relationship. The relationship between Student and College is many to one as a college can have many students however a student cannot study in multiple colleges at the same time. Student entity has attributes such as Stu_Id, Stu_Name & Stu_Addr and College entity has attributes such as Col_ID & Col_Name.

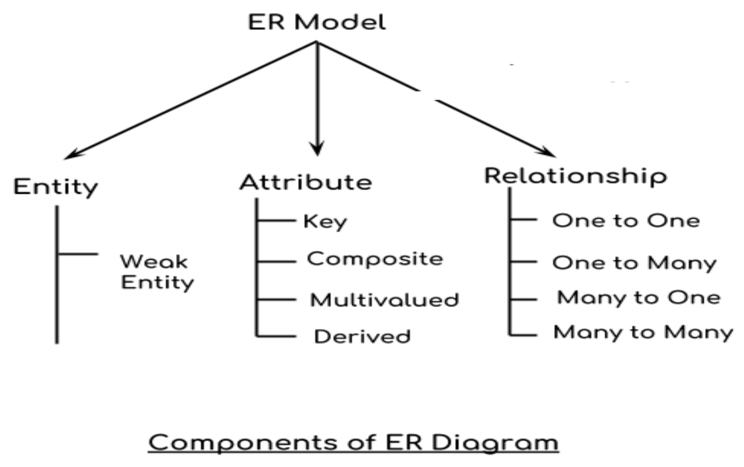


Sample E-R Diagram

Here are the geometric shapes and their meaning in an E-R Diagram. We will discuss these terms in detail in the next section(Components of a ER Diagram) of this guide so don't worry too much about these terms now, just go through them once.

Rectangle: Represents Entity sets.
Ellipses: Attributes
Diamonds: Relationship Set
Lines: They link attributes to Entity Sets and Entity sets to Relationship Set
Double Ellipses: Multi valued Attributes
Dashed Ellipses: Derived Attributes
Double Rectangles: Weak Entity Sets
Double Lines: Total participation of an entity in a relationship set

Compounds of a ER Diagram



As shown in the above diagram, an ER diagram has three main components:

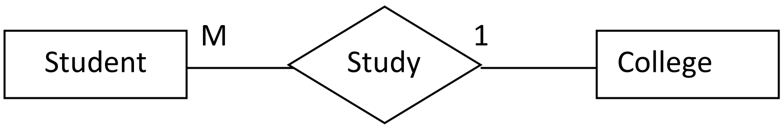
1. Entity
2. Attribute
3. Relationship

1. Entity

An entity is an object or component of data. An entity is represented as rectangle in an ER diagram.

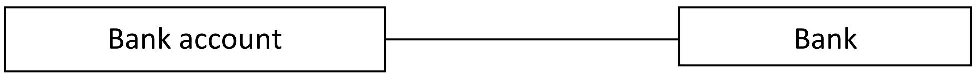
For example:

In the following ER diagram we have two entities Student and College and these two entities have many to one relationship as many students study in a single college. We will read more about relationships later, for now focus on entities.



Weak Entity:

An entity that cannot be uniquely identified by its own attributes and relies on the relationship with other entity is called weak entity. The weak entity is represented by a double rectangle. For example – a bank account cannot be uniquely identified without knowing the bank to which the account belongs, so bank account is a weak entity.



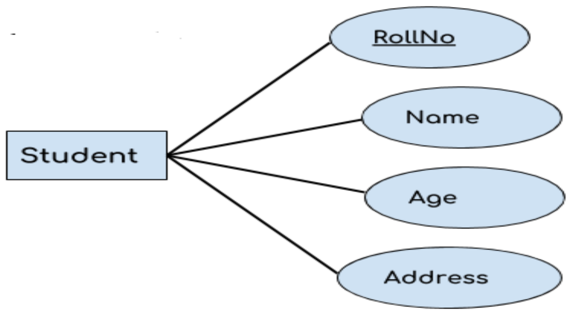
2. Attribute

An attribute describes the property of an entity. An attribute is represented as Oval in an ER diagram. There are four types of attributes:

a) Key attribute
b) Composite attribute
c) Multivalued attribute
d) Derived attribute

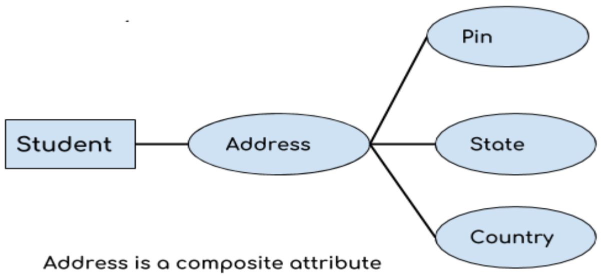
a) Key attribute:

A key attribute can uniquely identify an entity from an entity set. For example, student roll number can uniquely identify a student from a set of students. Key attribute is represented by oval same as other attributes however the text of key attribute is underlined.



b) Composite attribute:

An attribute that is a combination of other attributes is known as composite attribute. For example, In student entity, the student address is a composite attribute as an address is composed of other attributes such as pincode, state, country.



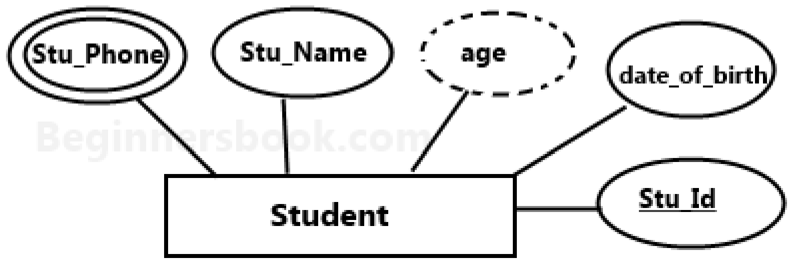
c) Multivalued attribute:

An attribute that can hold multiple values is known as multivalued attribute. It is represented with double ovals in an ER Diagram. For example – A person can have more than one phone numbers so the phone number attribute is multivalued.

d) Derived attribute:

A derived attribute is one whose value is dynamic and derived from another attribute. It is represented by dashed oval in an ER Diagram. For example – Person age is a derived attribute as it changes over time and can be derived from another attribute (Date of birth).

E-R diagram with multivalued and derived attributes:



3. Relationship

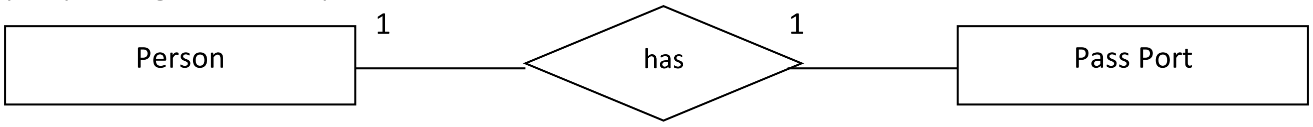
A relationship is represented by diamond shape in ER diagram, it shows the relationship among entities.

There are four types of relationships:

1. One to One
2. One to Many
3. Many to One
4. Many to Many

1. One to One Relationship

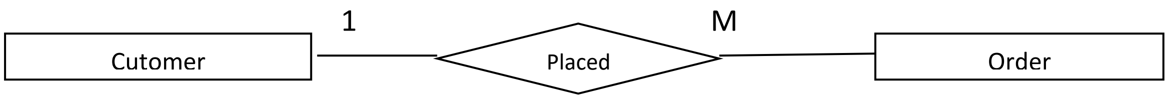
When a single instance of an entity is associated with a single instance of another entity then it is called one to one relationship. For example, a person has only one passport and a passport is given to one person.



Beginnerbook.com

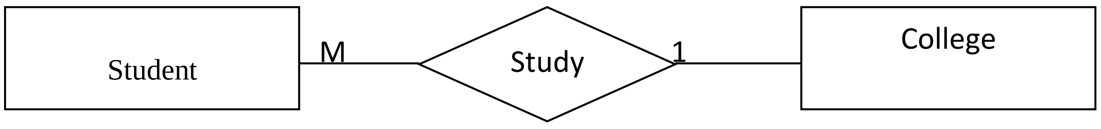
2. One to Many Relationship

When a single instance of an entity is associated with more than one instances of another entity then it is called one to many relationship. For example – a customer can place many orders but a order cannot be placed by many customers.



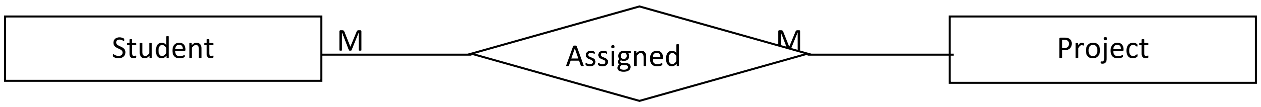
3. Many to One Relationship

When more than one instances of an entity is associated with a single instance of another entity then it is called many to one relationship. For example – many students can study in a single college but a student cannot study in many colleges at the same time.



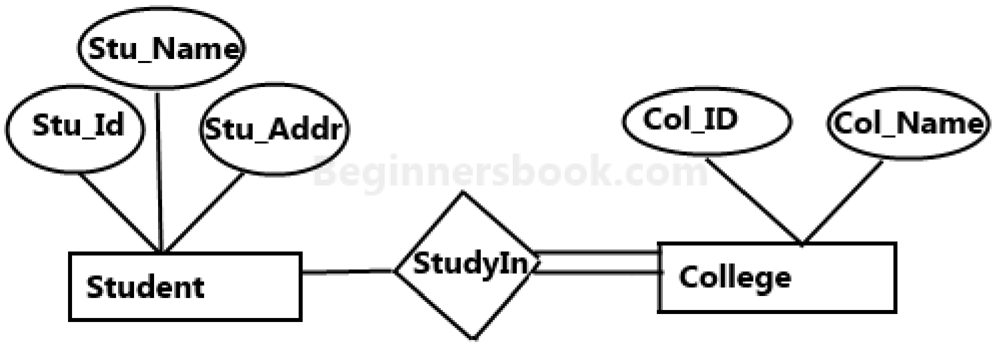
4. Many to Many Relationship

When more than one instances of an entity is associated with more than one instances of another entity then it is called many to many relationship. For example, a can be assigned to many projects and a project can be assigned to many students.



Total Participation of an Entity set

A Total participation of an entity set represents that each entity in entity set must have at least one relationship in a relationship set. For example: In the below diagram each college must have at-least one associated Student.



E-R Digram with total participation of College entity set in StudyIn relationship Set - This indicates that each college must have atleast one associated Student.

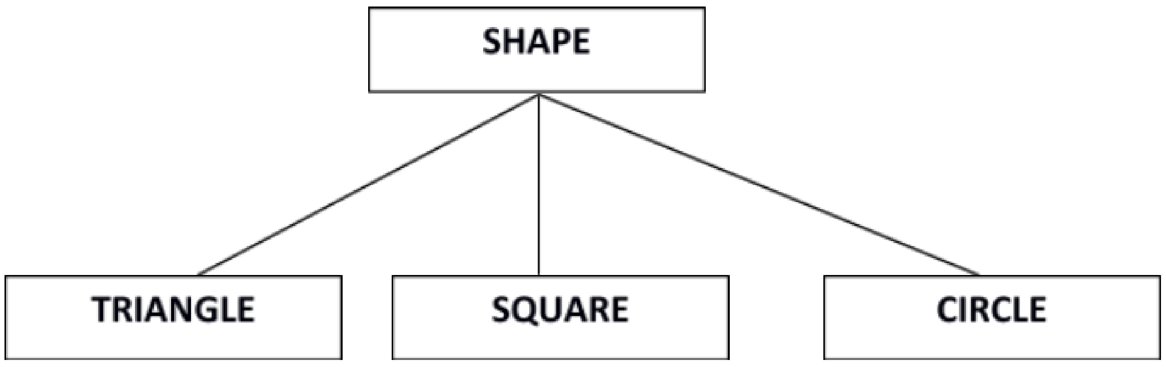
Extended Entity - Relationship (EE-R) Model

Incorporate the extensions to the original ER model. Enhanced ERD are high level models that represent the requirements and complexities of complex database.

In addition to ER model concepts EE-R includes:

a. Subclasses and Super classes.
b. Specialization and Generalization.
c. Category or union type.
d. Aggregation.

These concepts are used to create EE-R diagrams.
Subclasses and Super class
Super class is an entity that can be divided into further subtype.
For example – consider Shape super class.

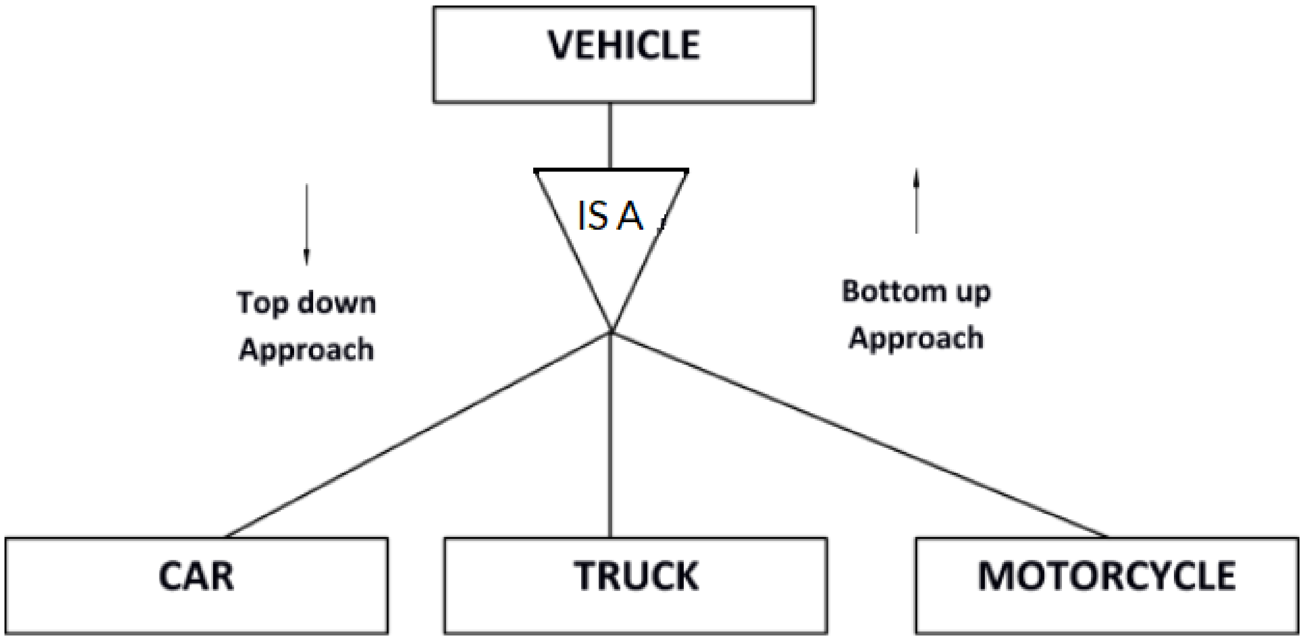


Super class shape has sub groups: Triangle, Square and Circle.

Sub classes are the group of entities with some unique attributes. Sub class inherits the properties and attributes from super class.

Specialization and Generalization:

Generalization is a process of generalizing an entity which contains generalized attributes or properties of generalized entities.

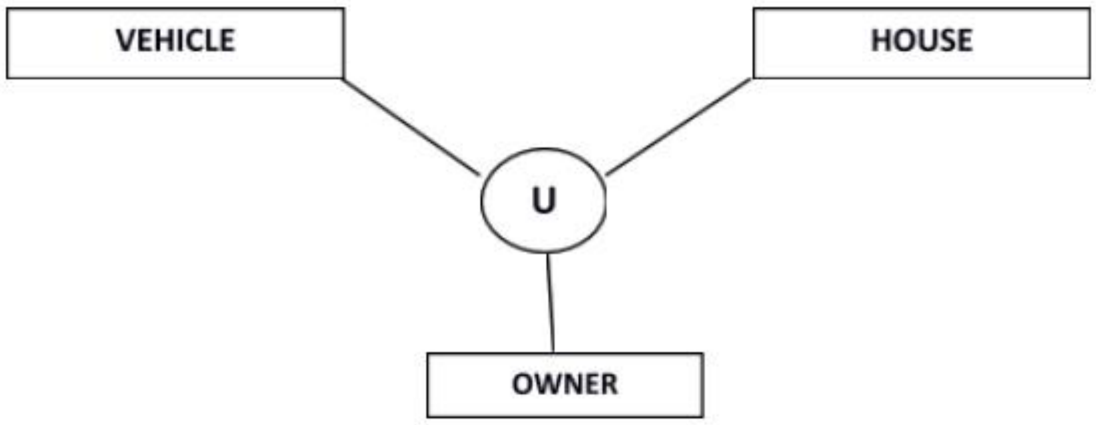


It is a Bottom up process i.e. considers we have 3 sub entities Car, Truck and Motorcycle. Now these three entities can be generalized into one super class named as Vehicle. Specialization is a process of identifying subsets of an entity that share some different characteristic. It is a top down approach in which one entity is broken down into low level entity.

In above example Vehicle entity can be a Car, Truck or Motorcycle.

Category or Union

Relationship of one super or sub class with more than one super class.

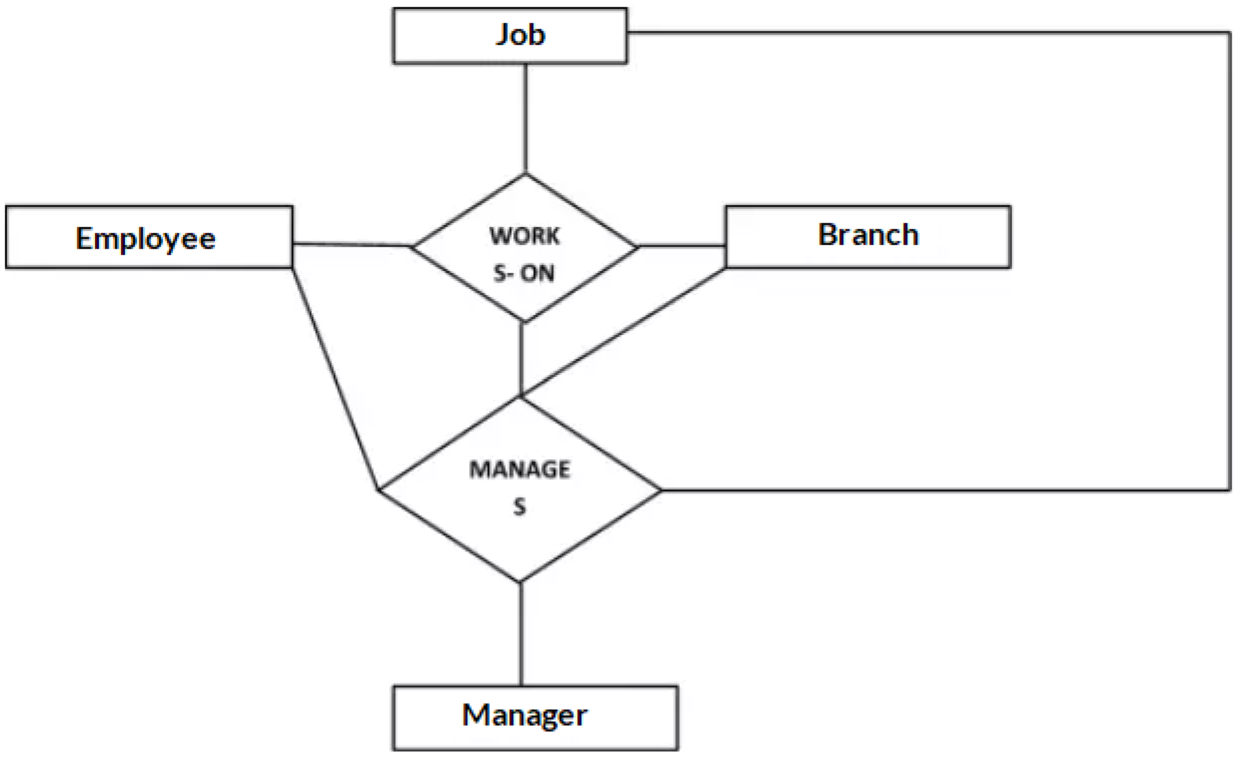


Owner is the subset of two super class:

Vehicle and House.

Aggregation

Represents relationship between a whole object and its component



Consider a ternary relationship Works_On between Employee, Branch and Manager. Now the best way to model this situation is to use aggregation, So, the relationship-set, Works_On is a higher level entity-set. Such an entity-set is treated in the same manner as any other entity-set. We can create a binary relationship, Manager, between Works_On and Manager to represent who manages what tasks.

Normalization of Database

Database Normalization is a technique of organizing the data in the database. Normalization is a systematic approach of decomposing tables to eliminate data redundancy (repetition) and undesirable characteristics like Insertion, Update and Deletion Anomalies. It is a multi-step process that puts data into tabular form, removing duplicated data from the relation tables.

Normalization is used for mainly two purposes,
Eliminating redundant (useless) data
Ensuring data dependencies make sense i.e data is logically stored.
Problems without Normalization

If a table is not properly normalized and has data redundancy then it will not only eat up extra memory space but will also make it difficult to handle and update the database, without facing data loss. Insertion, Updation and Deletion Anomalies are very frequent if database is not normalized. To understand these anomalies let us take an example of a Student table.

Rollno	Name	Branch	Hod	office_tel
401	Akon	CSE	Mr. X	53337
402	Bkon	CSE	Mr. X	53337
403	Ckon	CSE	Mr. X	53337

404	Dkon	CSE	Mr. X	53337

In the table above, we have data of 4 Computer Sci. students. As we can see, data for the fields branch, hod(Head of Department) and office_tel is repeated for the students who are in the same branch in the college, this is Data Redundancy.

Insertion Anomaly

Suppose for a new admission, until and unless a student opts for a branch, data of the student cannot be inserted, or else we will have to set the branch information as NULL.
Also, if we have to insert data of 100 students of same branch, then the branch information will be repeated for all those 100 students.

These scenarios are nothing but Insertion anomalies.

Updation Anomaly

What if Mr. X leaves the college? or is no longer the HOD of computer science department? In that case all the student records will have to be updated, and if by mistake we miss any record, it will lead to data inconsistency. This is Updation anomaly.

Deletion Anomaly

In our Student table, two different information's are kept together, Student information and Branch information. Hence, at the end of the academic year, if student records are deleted, we will also lose the branch information. This is Deletion anomaly.

Normalization Rule

Normalization rules are divided into the following normal forms:
First Normal Form
Second Normal Form
Third Normal Form
BCNF

First Normal Form (1NF)

For a table to be in the First Normal Form, it should follow the following 4 rules:
It should only have single(atomic) valued attributes/columns.
Values stored in a column should be of the same domain
All the columns in a table should have unique names.
And the order in which data is stored, does not matter.
In the next tutorial, we will discuss about the First Normal Form in details.

Second Normal Form (2NF)

For a table to be in the Second Normal Form,
It should be in the First Normal form.
And, it should not have Partial Dependency.
To understand what is Partial Dependency and how to normalize a table to 2nd normal for, jump to the Second Normal Form tutorial.

Third Normal Form (3NF)

A table is said to be in the Third Normal Form when,
It is in the Second Normal form.
And, it doesn't have Transitive Dependency.
Here is the Third Normal Form tutorial. But we suggest you to first study about the second

normal form and then head over to the third normal form.

Boyce and Codd Normal Form (BCNF)

Boyce and Codd Normal Form is a higher version of the Third Normal form. This form deals with certain type of anomaly that is not handled by 3NF. A 3NF table which does not have multiple overlapping candidate keys is said to be in BCNF. For a table to be in BCNF, following conditions must be satisfied:
R must be in 3rd Normal Form
and, for each functional dependency (X → Y), X should be a super Key.

Fourth Normal Form (4NF)

A table is said to be in the Fourth Normal Form when,
It is in the Boyce-Codd Normal Form.
And, it doesn't have Multi-Valued Dependency.

UNIT - V
IMPLEMENTATION USING ORACLE

Oracle CREATE TABLE
In Oracle, CREATE TABLE statement is used to create a new table in the database.
To create a table, you have to name that table and define its columns and datatype for each column.

Syntax:

1. **CREATE TABLE** table_name
2. (
3. column1 datatype [NULL | NOT NULL],
4. column2 datatype [NULL | NOT NULL],
5. ...
6. column_n datatype [NULL | NOT NULL]
7.);

Parameters used in syntax
1. table_name: It specifies the name of the table which you want to create.
2. column1, column2, ... column n: It specifies the columns which you want to add in the table. Every column must have a data type. Every column should either be defined as "NULL" or "NOT NULL". In the case, the value is left blank; it is treated as "NULL" as default.
Oracle CREATE TABLE Example
Here we are creating a table named customers. This table doesn't have any primary key.

1. **CREATE TABLE** customers
2. (customer_id number(10) NOT NULL,
3. customer_name varchar2(50) NOT NULL,
4. city varchar2(50)
5.);

This table contains three columns

a. **customer_id:**
It is the first column created as a number data type (maximum 10 digits in length) and cannot contain null values.
b. **customer_name:**
It is the second column created as a varchar2 data type (50 maximum characters in length) and cannot contain null values.
c. **City:**
This is the third column created as a varchar2 data type. It can contain null values.

Oracle CREATE TABLE Example with primary key

1. **CREATE TABLE** customers
2. (customer_id number(10) NOT NULL,
3. customer_name varchar2(50) NOT NULL,
4. city varchar2(50),
5. **CONSTRAINT** customers_pk **PRIMARY KEY** (customer_id)
6.);

What is Primary key?

A primary key is a single field or combination of fields that contains a unique record. It must be filled. None of the field of primary key can contain a null value. A table can have only one primary key.

Oracle ALTER TABLE Statement

In Oracle, ALTER TABLE statement specifies how to add, modify, drop or delete columns in a table. It is also used to rename a table.

How to add column in a table

Syntax:

1. **ALTER TABLE** table_name
2. **ADD** column_name **column**-definition;

Example:

Consider that already existing table customers. Now, add a new column customer_age into the table customers.

1. **ALTER TABLE** customers
2. **ADD** customer_age varchar2(50);

Now, a new column "customer_age" will be added in customers table.

How to add multiple columns in the existing table

Syntax:

1. **ALTER TABLE** table_name
2. **ADD** (column_1 **column**-definition,
3. column_2 **column**-definition,
4. ...
5. column_n column_definition);

Example

1. **ALTER TABLE** customers
2. **ADD** (customer_type varchar2(50),
3. customer_address varchar2(50));

Now, two columns customer_type and customer_address will be added in the table customers.

How to modify column of a table

Syntax:

1. **ALTER TABLE** table_name
2. **MODIFY** column_name column_type;

Example:

1. **ALTER TABLE** customers
2. **MODIFY** customer_name varchar2(100) not null;

Now the column column_name in the customers table is modified
to varchar2 (100) and forced the column to not allow null values.

How to modify multiple columns of a table

Syntax:

1. **ALTER TABLE** table_name
2. **MODIFY** (column_1 column_type,

3. column_2 column_type,
4. ...
5. column_n column_type);

Example:

1. **ALTER TABLE** customers
2. **MODIFY** (customer_name varchar2(100) not null,
3. city varchar2(100));

This will modify both the customer_name and city columns in the table.
How to drop column of a table

Syntax:

1. **ALTER TABLE** table_name
2. **DROP COLUMN** column_name;

Example:

1. **ALTER TABLE** customers
2. **DROP COLUMN** customer_name;

This will drop the customer_name column from the table.
How to rename column of a table

Syntax:

1. **ALTER TABLE** table_name
2. RENAME **COLUMN** old_name **to** new_name;

Example:

1. **ALTER TABLE** customers
2. RENAME **COLUMN** customer_name **to** cname;

This will rename the column customer_name into cname.
How to rename table

Syntax:

1. **ALTER TABLE** table_name
2. RENAME **TO** new_table_name;

Example:

1. **ALTER TABLE** customers
2. RENAME **TO** retailers;

This will rename the customer table into "retailers" table.

Introduction to Oracle CREATE SEQUENCE statement
The CREATE SEQUENCE statement allows you to create a new sequence in the database.
Here is the basic syntax of the CREATE SEQUENCE statement:

```
1 CREATE SEQUENCE schema_name.sequence_name
2 [INCREMENT BY interval]
3 [START WITH first_number]
4 [MAXVALUE max_value | NOMAXVALUE]
5 [MINVALUE min_value | NOMINVALUE]
```

```
6 [CYCLE | NOCYCLE]
7 [CACHE cache_size | NOCACHE]
8 [ORDER | NOORDER];
```

CREATE SEQUENCE
Specify the name of the sequence after the CREATE SEQUENCE keywords. If you want to create a sequence in a specific schema, you can specify the schema name in along with the sequence name.

INCREMENT BY
Specify the interval between sequence numbers after the INCREMENT BY keyword.
The interval can have less than 28 digits. It also must be less than MAXVALUE - MINVALUE.
If the interval is positive, the sequence is ascending e.g., 1,2,3,…
If the interval is negative, the sequence is descending e.g., -1, -2, -3 …
The default value of interval is 1.

START WITH
Specify the first number in the sequence.
The default value of the first number is the minimum value of the sequence for an ascending sequence and maximum value of the sequence for a descending sequence.

MAXVALUE
Specify the maximum value of the sequence.
The max_value must be equal to or greater than first_number specify after the START WITH keywords.

NOMAXVALUE
Use NOMAXVALUE to denote a maximum value of 10^27 for an ascending sequence or -1 for a descending sequence. Oracle uses this option as the default.
MINVALUE
Specify the minimum value of the sequence.
The min_value must be less than or equal to the first_number and must be less than max_value.

NOMINVALUE
Use NOMINVALUE to indicate a minimum value of 1 for an ascending sequence or -10^26 for a descending sequence. This is the default.

CYCLE
Use CYCLE to allow the sequence to generate value after it reaches the limit, min value for a descending sequence and max value for an ascending sequence.
When an ascending sequence reaches its maximum value, it generates the minimum value.
On the other hand, when a descending sequence reaches its minimum value, it generates the maximum value.

NOCYCLE
Use NOCYCLE if you want the sequence to stop generating the next value when it reaches its limit. This is the default.

CACHE

Specify the number of sequence values that Oracle will preallocate and keep in the memory for faster access.

The minimum of the cache size is 2. The maximum value of the cache size is based on this formula:

```
(CEIL (MAXVALUE - MINVALUE)) / ABS (INCREMENT)
```

In case of a system failure event, you will lose all cached sequence values that have not been used in committed SQL statements.

ORDER

Use ORDER to ensure that Oracle will generate the sequence numbers in order of request. This option is useful if you are using Oracle Real Application Clusters. When you are using exclusive mode, then Oracle will always generate sequence numbers in order.

NOORDER

Use NOORDER if you do not want to ensure Oracle to generate sequence numbers in order of request. This option is the default.

Oracle CREATE SEQUENCE statement examples

The following statement creates an ascending sequence called id_seq, starting from 10, incrementing by 10, minimum value 10, maximum value 100. The sequence returns 10 once it reaches 100 because of the CYCLE option.

```
CREATE SEQUENCE id_seq
    INCREMENT BY 10
    START WITH 10
    MINVALUE 10
    MAXVALUE 100
    CYCLE
    CACHE 2;
```

To get the next value of the sequence, you use the NEXTVAL pseudo-column:

```
SELECT
 id_seq.NEXTVAL
FROM
 dual;
```

Here is the output:

```
 NEXTVAL
----------
  10
```

To get the current value of the sequence, you use the CURRVAL pseudo-column:

```
SELECT
    id_seq.CURRVAL
FROM
    dual;
```

The current value is 10:

```
    CURRVAL
----------
```

```
       10
```

This SELECT statement uses the id_seq.NEXTVAL value repeatedly:

```
SELECT
    id_seq.NEXTVAL
FROM
    dual
CONNECT BY level <= 9;
```

Here is the output:

```
    NEXTVAL
----------
        20
        30
        40
        50
        60
        70
        80
        90
       100

9 rows selected
```

Because we set the CYCLE option for the id_seq sequence, the next value of the id_seq will be 10:

```
SELECT id_seq.NEXTVAL FROM dual;
```

And here is the output:

```
   NEXTVAL
----------
       10
```

Oracle Procedures

A procedure is a group of PL/SQL statements that can be called by name. The call specification (sometimes called call spec) specifies a java method or a third-generation language routine so that it can be called from SQL and PL/SQL.

Create Procedure

Syntax

```
CREATE [OR REPLACE] PROCEDURE procedure_name
 [ (parameter [,parameter]) ]
IS
[declaration_section]
BEGIN
executable_section
[EXCEPTION
exception_section]
END [procedure_name];
```

Following are the three types of procedures that must be defined to create a procedure.

1. **IN:** It is a default parameter. It passes the value to the subprogram.
2. **OUT:** It must be specified. It returns a value to the caller.
3. **IN OUT:** It must be specified. It passes an initial value to the subprogram and returns an updated value to the caller.

Oracle Create procedure example
In this example, we are going to insert record in the "user" table. So you need to create user table first.

Table creation:
create table user(id number(10) **primary key**,**name** varchar2(100));
Now write the procedure code to insert record in user table.

Procedure Code:

```
create or replace procedure "INSERTUSER"
(id IN NUMBER,
name IN VARCHAR2)
is
begin
insert into user values(id,name);
end;
/
```

Output:
Procedure created.
Oracle program to call procedure
Let's see the code to call above created procedure.

```
BEGIN
Insert user(101,'Rahul');
dbms_output.put_line('record inserted successfully');
END;
/
```

Now, see the "USER" table, you will see one record is inserted.

ID	Name
101	Rahul

Oracle Queries
You can execute many queries in oracle database such as insert, update, delete, alter table, drop, create and select.
Oracle Select Query
Oracle select query is used to fetch records from database. For example:

1. **SELECT** * **from** customers;
More Details...

2. Oracle Insert Query

Oracle insert query is used to insert records into table. For example:
insert into customers **values**(101,'rahul','delhi');
More Details...

3. Oracle Update Query
Oracle update query is used to update records of a table. For example:
update customers **set name**='bob', city='london' **where** id=101;
More Details...

4. Oracle Delete Query
Oracle update query is used to delete records of a table from database. For example:
delete from customers **where** id=101;

5. Oracle Truncate Query
Oracle update query is used to truncate or remove records of a table. It doesn't remove structure. For example:
truncate table customers;

6. Oracle Drop Query
Oracle drop query is used to drop a table or view. It doesn't have structure and data.
For example:
drop table customers;

7. Oracle Create Query
Oracle create query is used to create a table, view, sequence, procedure and function. For example:

1. **CREATE TABLE** customers
2. (id number(10) NOT NULL,
3. **name** varchar2(50) NOT NULL,
4. city varchar2(50),
5. **CONSTRAINT** customers_pk **PRIMARY KEY** (id)
6.);

8. Oracle Alter Query
Oracle alter query is used to add, modify, delete or drop colums of a table. Let's see a query to add column in customers table:

1. **ALTER TABLE** customers
2. **ADD** age varchar2(50);

Oracle Function

A function is a subprogram that is used to return a single value. You must declare and define a function before invoking it. It can be declared and defined at a same time or can be declared first and defined later in the same block.
CREATE function in Oracle

Syntax

1. **CREATE** [OR REPLACE] **FUNCTION** function_name
2. [(parameter [,parameter])]
3. **RETURN** return_datatype
4. **IS** | **AS**

```
[declaration_section]
BEGIN
executable_section
[EXCEPTION
exception_section]
END [function_name];
```

You must have define some parametrs before creating a procedure or a function. These parameters are

1. **IN:** It is a default parameter. It passes the value to the subprogram.
2. **OUT:** It must be specified. It returns a value to the caller.
3. **IN OUT:** It must be specified. It passes an initial value to the subprogram and returns an updated value to the caller.

Oracle Function Example

Let's see a simple example to **create a function**.

```
create or replace function adder(n1 in number, n2 in number)
return number
is
n3 number(8);
begin
n3 :=n1+n2;
return n3;
end;
/
```

Now write another program to **call the function**.

```
DECLARE
  n3 number(2);
BEGIN
  n3 := adder(11,22);
  dbms_output.put_line('Addition is: ' || n3);
END;
/
```

Output:

```
Addition is: 33
Statement processed.
0.05 seconds
```