RDBMS with SQL (BIT245)

Sulav Nepal

Email: nep.sulav@live.com

Contact No.: 9849194892

Master's in Computer Information System (MCIS) – Pokhara University
Bachelor of Science. Computer Science & Information Technology (B.Sc. CSIT) – Tribhuwan University
Microsoft Technology Associate (MTA): Windows Server Administration Fundamentals
Microsoft Certified Technology Specialist (MCTS): Windows Server 2008 R2, Server Virtualization
Microsoft Specialist (MS): Programming in HTML5 with JavaScript and CSS3
Microsoft Students Partner (MSP) 2012 for Nepal

Syllabus

- UNIT 1: Introduction
  - The SQL Language
  - The Role of SQL
  - SQL Success Factors
    - Official SQL Standards
    - Microsoft Support
    - Relational Foundation
    - Complete Database Language
    - Client/Server Architecture
    - Retrieving Data
    - Creating a Database

UNIT 1: Introduction

Introduction to SQL

- SQL is a database computer language designed for the retrieval and management of data in a relational database.
- SQL stands for Structured Query Language.
- SQL lets you access and manipulate databases.
- SQL became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987.

Why to learn SQL?

- SQL is a computer language for storing, manipulating and retrieving data stored in a relational database.
- SQL is the standard language for Relational Database System.
- All the RDBMS like MySQL, MS Access, Oracle, Sybase, Informix, Postgres and SQL Server use SQL as their standard database language.
- Also they are using different dialects (*a particular version of a programming language*), such as:
  - MS SQL Server using T-SQL
  - Oracle using PL/SQL
  - MS Access version of SQL is called JET SQL (native format) etc.

What can SQL do?

- SQL can execute queries against a database.
- SQL can retrieve data from a database.
- SQL can insert records in a database.
- SQL can update records in a database.
- SQL can delete records from a database.
- SQL can create new databases.
- SQL can create new tables in a database.
- SQL can create stored procedures in a database.
- SQL can create views in a database.
- SQL can set permissions on tables, procedures, and views.

Applications of SQL

- Allows users to access data in the RDBMS.
- Allows users to describe the data.
- Allows users to define the data in a database and manipulate that data.
- Allows to embed within other languages using SQL modules, libraries, & pre-compilers.
- Allows users to create and drop databases and tables.
- Allows users to create view, stored procedure, functions in a database.
- Allows users to set permissions on tables, procedures and views.

SQL – Creating Roles

- A role is created to ease setup and maintenance of the security mode.
- It is a named group of related privileges that can be granted to the user.
- When there are many users in a database it becomes difficult to grant or revoke privileges to users.
- Therefore, if you define roles:
  - You can grant or revoke privileges to users, thereby automatically granting or revoking privileges.
  - You can either create Roles or use the system roles pre-defined.

SQL – Creating Roles

- Some of the privileges granted to the system roles are:

System Roles	Privileges granted to the Role
Connect	Create table, Create view, Create synonym, Create sequence, Create session, etc.
Resource	Create Procedure, Create Sequence, Create Table, Create Trigger, etc. The primary usage of the Resource role is to restrict access to database objects.
DBA	All system privileges

SQL is a Standard – BUT . . . . .

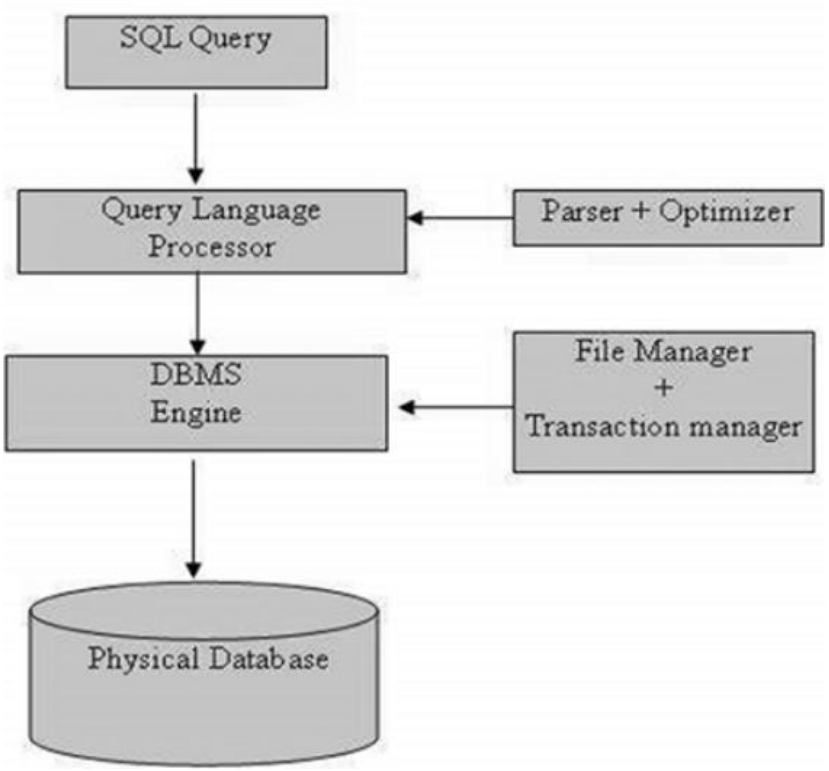
- Although SQL is an ANSI/ISO standard, there are different versions of the SQL language.
- However, to be compliant with the ANSI standard, they all support at least the major commands (such as SELECT, UPDATE, DELETE, INSERT, WHERE) in a similar manner.

History of SQL

- 1970 – Dr. Edgar F. "Ted" Codd of IBM is known as the father of relational databases. He described a relational model for databases.
- 1974 – Structured Query Language appeared.
- 1978 – IBM worked to develop Codd's ideas and released a product named System/R.
- 1986 – IBM developed the first prototype of relational database and standardized by ANSI. The first relational database was released by Relational Software which later came to be known as Oracle.

SQL Execution Process

- When you are executing an SQL command for any RDBMS, the system determines the best way to carry out your request and SQL engine figures out how to interpret the task.



SQL Commands

- The standard SQL commands to interact with relational databases are CREATE, SELECT, INSERT, UPDATE, DELETE, and DROP.
- These commands can be classified into the following groups based on their nature:
  - DDL – Data Definition Language
  - DML – Data Manipulation Language
  - DCL – Data Control Language
  - DQL – Data Query Language

DDL – Data Definition Language

S.N.	Command & Description
1	**CREATE** Creates a new table, a view of a table, or other object in the database.
2	**ALTER** Modifies an existing database object, such as a table.
3	**DROP** Deletes an entire table, a view of a table or other objects in the database.

DML – Data Manipulation Language

S.N.	Command & Description
1	**INSERT** Creates a record.
2	**UPDATE** Modifies records.
3	**DELETE** Deletes records.

DCL – Data Control Language

S.N.	Command & Description
1	**GRANT** Gives a privilege to user.
2	**REVOKE** Takes back privileges granted from user.

DQL – Data Query Language

S.N.	Command & Description
1	**SELECT** Retrieves certain records from one or more tables.

SQL Standards

- Many Relational Database vendors such as MS SQL Server, MySQL, Oracle, and DB2 offer different versions of SQL.
- Because of this, ANSI develops a standard for SQL language which runs across all databases.
- ANSI SQL is the standard for SQL language that is defined by ANSI (American National Standards Institute).
- When we talk about SQL in general, we usually refer to this ANSI standard SQL.
- The first version of ANSI SQL was formalized in 1986.
- Since then, the standard has been updated several times.

SQL Standards

- **SQL – 86**
  - Year: 1986
  - Alias: SQL – 87
  - This is the first version standardized by ANSI.
- **SQL – 89**
  - Year: 1989
  - This version includes minor revisions that add integrity constraints.
- **SQL – 92**
  - Year: 1992
  - Alias: SQL2
  - This version includes major revisions on the SQL language.
  - Many database systems use this standard for their specification language.

SQL Standards

- **SQL:1999**
  - Year: 1999
  - Alias: SQL3
  - This version introduces many new features, such as regex matching, triggers, some object-oriented features, and OLAP capability.
  - SQL:1999 also adds BOOLEAN data type, but many commercial database servers do not support it as a column type.
  - In the meanwhile, SQL:1999 is deprecated.
- **SQL:2003**
  - Year: 2003
  - This version makes minor modifications to all parts of SQL:1999.
  - This SQL:2003 version also introduces new features such as:
    - Window functions, which are very useful for data analysis.
    - Columns with auto-generated value and identity specification.

SQL Standards

- **SQL:2006**
  - Year: 2006
  - This version defines ways of importing, storing, and manipulating XML data.
  - Additionally, it lets applications use Xquery to query data in XML format.
- **SQL:2008**
  - Year: 2008
  - This version includes minor revisions that add integrity constraints.

RDBMS

- RDBMS stands for Relational Database Management System.
- RDBMS is the basis for SQL, and for all modern database systems like MS SQL Server, IBM DB2, Oracle, MySQL, and Microsoft Access.
- RDBMS is a database management system that is based on the relational model as introduced by E.F. Codd.

Data Integrity

- The following categories of data integrity exist with each RDBMS
- Entity Integrity
  - There are no duplicate rows in a table.
- Domain Integrity
  - Enforces valid entries for a given column by restricting the type, the format, or the range of values.
- Referential Integrity
  - Rows cannot be deleted, which are used by other records.
- User-Defined Integrity
  - Enforces some specific business rules that do not fall into entity, domain or referential integrity.

Table

- The data in RDBMS is stored in database objects called tables.
- A table is a collection of related data entries and it consists of columns and rows.
- The following program is an example of a CUSTOMERS table:

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

Field

- Every table is broken up into smaller entities called fields.
- A field is a column in a table that is designed to maintain specific information about every record in the table.
- The fields in the CUSTOMERS table consist of ID, NAME, AGE, ADDRESS, and SALARY.

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

Record (or Row)

- A record (is also called as a row of data) is each individual entry that exists in a table.
- For example, there are 7 records in the above CUSTOMERS table.
- Following is a single row of data or record in CUSTOMERS table.
- A record is a horizontal entity in a table.

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

Column

- A column is a vertical entity in a table that contains all information associated with a specific field in a table.
- For example, a column in the CUSTOMERS table is ADDRESS, which represents location description and would be as shown below:

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

NULL Value

- A NULL value in a table is a value in a field that appears to be blank, which means a field with a NULL value is a field with no value.
- It is very important to understand that a NULL value is different than a zero value or a field that contains spaces.
- A field with a NULL value is the one that has been left blank during a record creation.

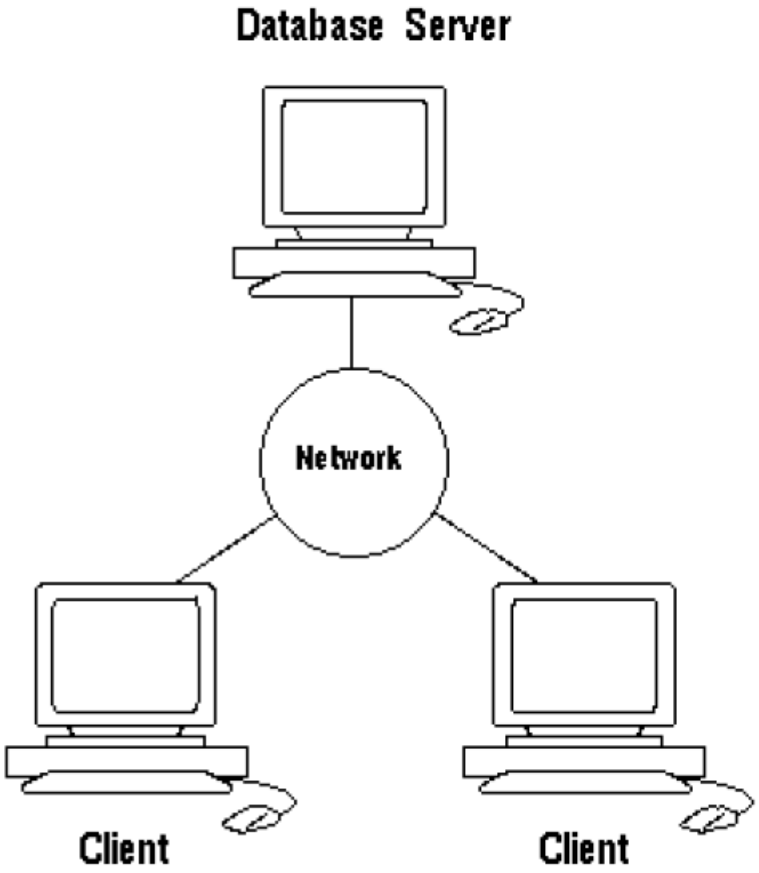
Complete Database Language

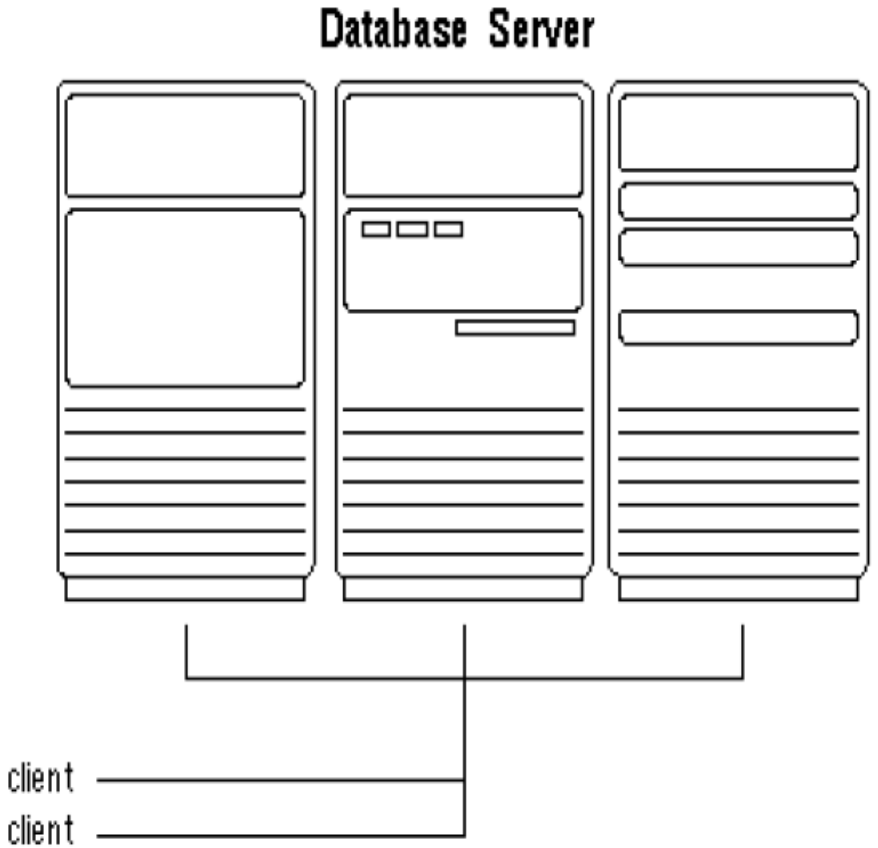
- SQL is a database language designed for managing data held in a relational database management system.
- SQL was initially developed by IBM in the early 1970s.
- In a DBMS, the SQL database language is used to: Create the database and table structures.
- There are large number of database languages like: Oracle, MySQL, MS Access, dBase, FoxPro, etc.
- SQL statements commonly used in Oracle and MS Access can be categorized as Data Definition Language (DDL), Data Control Language (DCL), and Data Manipulation Language (DML).

Client-Server Architecture

- An architecture of a computer network in which many clients (remote processors) request and receive service from a centralized server (host computer).
- Client computers provide an interface to allow a computer user to request services of the server and to display the results the server returns.
- In order to retrieve the desired data the user present a set of criteria by a query.
- Then the DBMS, software for managing databases, selects the demanded data from the database.
- The retrieved data may be stored in a file, printed, or viewed on the screen.

Client-Server Architecture





Retrieving a Data

- In order to retrieve the desired data the user present a set of criteria by a query.
- Then the DBMS, software for managing databases, selects the demanded data from the database.
- The retrieved data may be stored in a file, printed, or viewed on the screen.
- For example:

SELECT

column_name1, column_name2, . . . , column_nameN

FROM

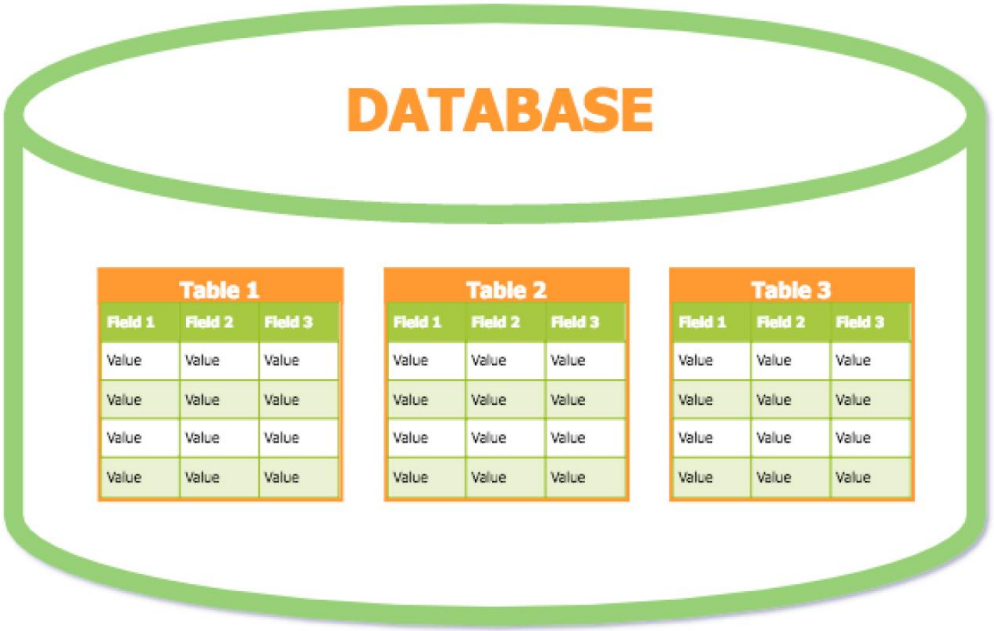
table_name;

Creating a Database

- The following statement is an example of a CREATE DATABASE statement:

  CREATE DATABASE *database_name*;

- Always the database name should be unique within the RDBMS.



END OF UNIT ONE

Syllabus

- UNIT 2: Relational Databases
  - Early Data Models
    - File Management Systems
    - Hierarchical Databases
    - Network Databases
  - The Relational Data Model
    - The Sample Database
    - Tables
    - Primary Keys
    - Relationships
    - Foreign Keys
  - Codd's 12 Rules for Relational Databases

UNIT 2: Relational Databases

Early Data Models

- A database model is a type of data model that determines the logical structure of a database and fundamentally determines in which manner data can be stored, organized, and manipulated.
- The early data models are:
  - File Management System
  - Hierarchical Model
  - Network Model

File Management System

- A file management system is a type of software that manages data files in a computer system.
- It has limited capabilities and is designed to manage individual or group files, such as special office documents and records.
- In this system, the user has to write the procedures for managing the database.
- File system provides the detail of the data representation and storage of data.

File Management System

- File system doesn't have a crash mechanism, i.e. if the system crashes while entering some data, then the content of the file will lost.
- It is very difficult to protect a file under the file system.
- File system can't efficiently store and retrieve the data.
- In the file system, concurrent access has many problems like redirecting the file while other deleting some information or updating some information.

File Management System Advantages

- **Data Security**
  - Security advantages over electronic filing.
- **Complexity**
  - Less complex than electronic systems, which can make it easier for untrained people to access and manipulate data.
- **Access Time**
  - Can be accessed in a very quick time.
- **Simple to Use**
  - Very simple to set up and use
- **Minimal Investment**
  - No need of investment in software database that may include an upfront purchase and ongoing licensing fees.

File Management System Disadvantages

- **Data Redundancy**
  - Since each application has its own data file, the same data may have to be recorded and stored in many files.
  - For example, personal file and payroll file, both contain data on employee name.
- **Data Inconsistency**
  - Data redundancy leads to data inconsistency especially when data is to be updated.
  - Data inconsistency occurs due to the same data items that appear in more than one file do not get updated simultaneously in each and every file.
- **Program Dependence**
  - The reports produced by the file management system are program dependent, which means if any change in the format or structure of data and records in the file is to be made, the programs have to be modified similarly.
  - Also, a new program will have to be developed to produce a new report.
- **Limited Data Sharing**
  - There is limited data sharing possibilities with the traditional file system.
  - Each application has its own private files and users have little choice to share the data outside their own applications.

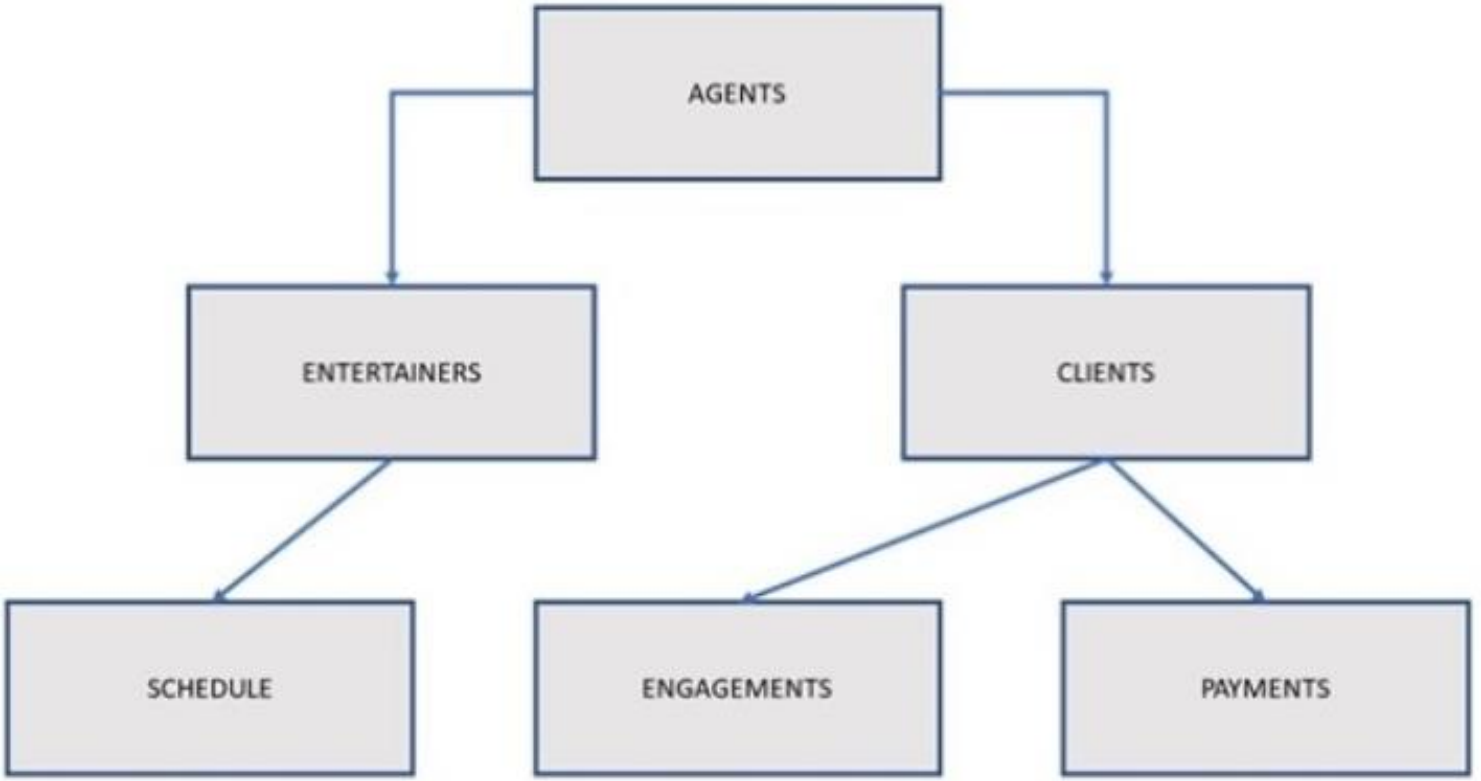
Hierarchical Model

- A hierarchical model represents the data in a tree-like structure in which there is a single parent for each record.
- To maintain order there is a sort field which keeps sibling nodes into a recorded manner.
- These type of models are designed basically for the early mainframe DBMS, like the Information Management System (IMS) by IBM.
- This model structure allows the one-to-one and a one-to-many relationship between two/various types of data.
- This structure is very helpful in describing many relationships in the real world; table of contents, any nested and sorted information.

Hierarchical Model

- The hierarchical structure is used as the physical order of records in storage.
- One can access the records by navigating down through the data structure using pointers which are combined with sequential accessing.
- Therefore, the hierarchical structure is not suitable for certain database operations when a full path is not also included for each record.
- Data in this type of database is structured hierarchically and is typically developed as an inverted tree.
- The "root" in the structure is a single table in the database and other tables act as the branches flowing from the root.

Hierarchical Model



Hierarchical Model – Advantages

- A user can retrieve data very quickly due to the presence of explicit links between the table structures.
- The referential integrity is built in and automatically enforced due to which a record in a child table must be linked to an existing record in a parent table.
- If a record is deleted in the parent table then that will cause all associated records in the child table to be deleted as well.

Hierarchical Model–Disadvantages

- When a user needs to store a record in a child table that is currently unrelated to any record in a parent table, it gets difficulty in recording and user must record an additional entry in the parent table.
- This type of database cannot support complex relationships, and there is also a problem of redundancy, which can result in producing inaccurate information due to the inconsistent recording of data at various sites.

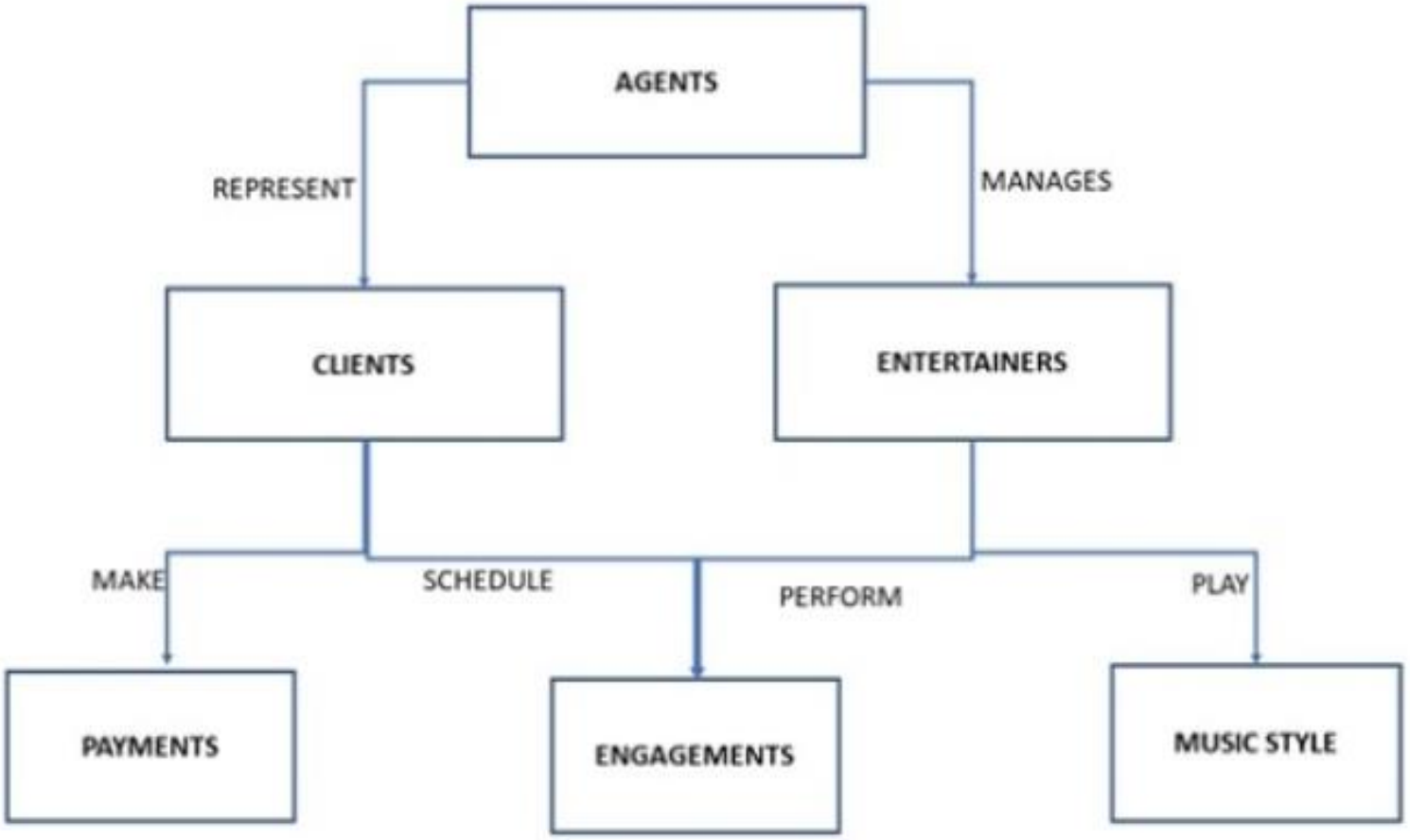
Network Model

- The network model is the extension of the hierarchical structure because it allows many-to-many relationships to be managed in a tree-like structure that allows multiple parents.
- There are two fundamental concepts of a network model:
  - Records contain fields which need hierarchical organization.
  - Sets are used to define one-to-many relationships between records that contain one owner, many members.
- A record may act as an owner in any number of sets, and a member in any number of sets.
- **NOTE:** Set must not be confused with the mathematical set.

Network Model

- The network model has the following major features:
  - It can represent redundancy in data more efficiently that that in the hierarchical model.
  - There can be more than one path from a previous node to successor node/s.
  - The operations of the network model are maintained by indexing structure of linked list (circular) where a program maintains a current position and navigates from one record to another by following the relationships in which the record participates.
  - Records can also be located by supplying key values.

Network Model



Network Model – Advantages

- Fast data access.
- It also allows users to create queries that are more complex than those they created using a hierarchical database. So, a variety of queries can be run over this model.

Network Model – Disadvantages

- A user must be very familiar with the structure of the database to work through the set structures.
- Updating inside this database is a tedious task.
- One cannot change a set structure without affecting the application programs that use this structure to navigate through the data.
- If you change a set structure, you must also modify all references made from within the application program to that structure.

Network vs. Hierarchical Model

Network Database Model	Hierarchical Database Model
Many-to-many relationship	One-to-many relationship
Easily accessed because of the linkage between the information	Difficult to navigate because of its strict owner to member connection
Great flexibility among the information files because the multiple relationships among the files	Less flexibility with the collection of information because of the hierarchical position of the files

Network vs. Relational Model

Network Database Model	Relational Database Model
The files are greatly related	Information is stored on separate tables tied together with other clumps of information

Relational Database Model

- The relational model represents the database as a collection of relations.
- A relation is nothing but a table of values.
- Every row in the table represents a collection of related data values.
- These rows in the table denote a real-world entity or relationship.
- The table name and column names are helpful to interpret the meaning of values in each row.

Relational Database Model

- The data are represented as a set of relations.
- In the relational model, data are stored as tables.
- However, the physical storage of the data is independent of the way the data are logically organized.
- Some popular Relational Database Management Systems (RDBMS) are:
  - DB2 and Informix Dynamic Server – IBM
  - Oracle and RDB – Oracle
  - SQL Server and Access – Microsoft

Relational Model Concepts

- **Attribute**
  - Each column in a Table.
  - Attributes are the properties which define a relation.
  - For example: Student_Rollno, NAME, etc.
- **Tables**
  - In the Relational model the relations are saved in the table format.
  - It is stored along with its entities.
  - A table has two properties rows and columns.
  - Rows represent records and columns represent attributes.
- **Tuple**
  - It is nothing but a single row of a table, which contains a single record.

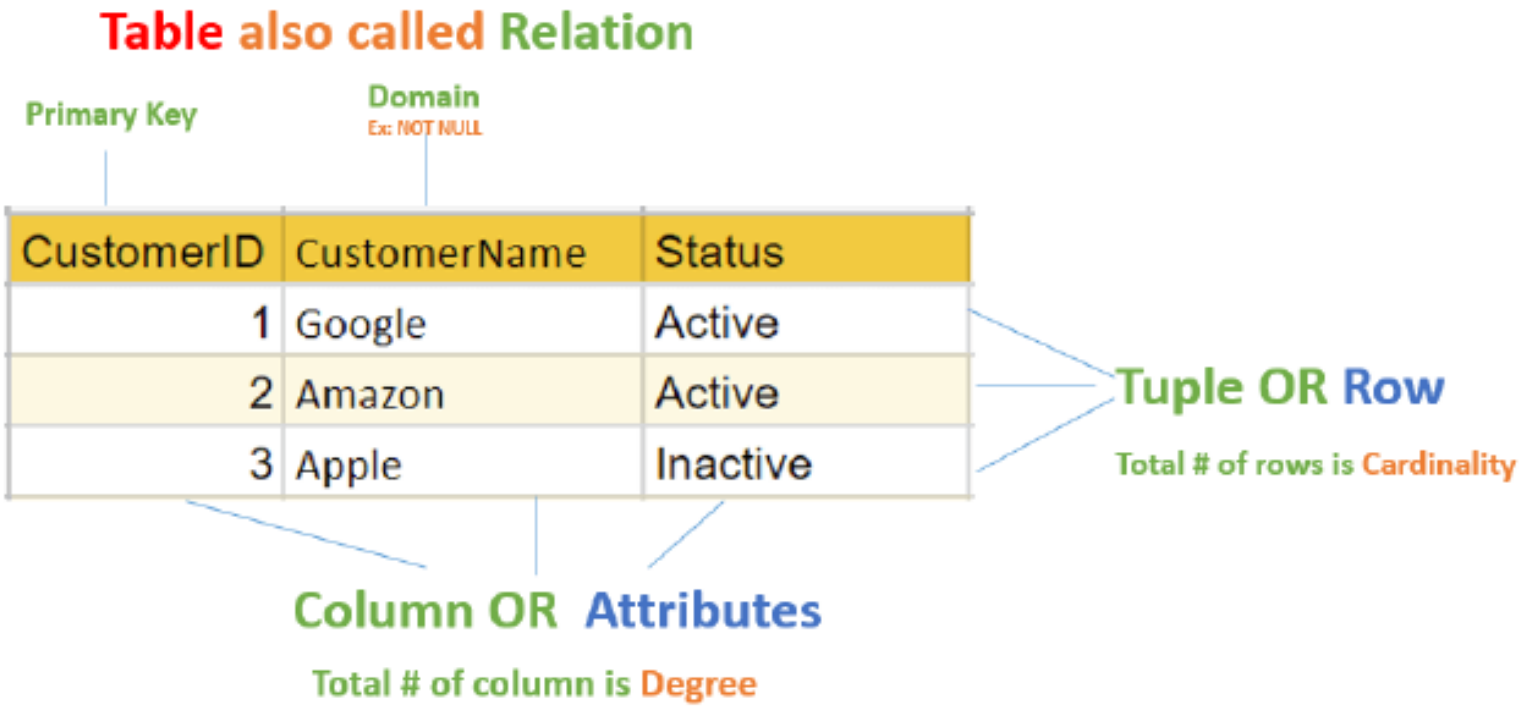
Relational Model Concepts

- **Relation Schema**
  - A relation schema represents the name of the relation with its attributes.
- **Degree**
  - The total number of attributes which in the relation is called the degree of the relation.
- **Cardinality**
  - Total number of rows present in the Table.
- **Column**
  - The column represents the set of values for a specific attribute.

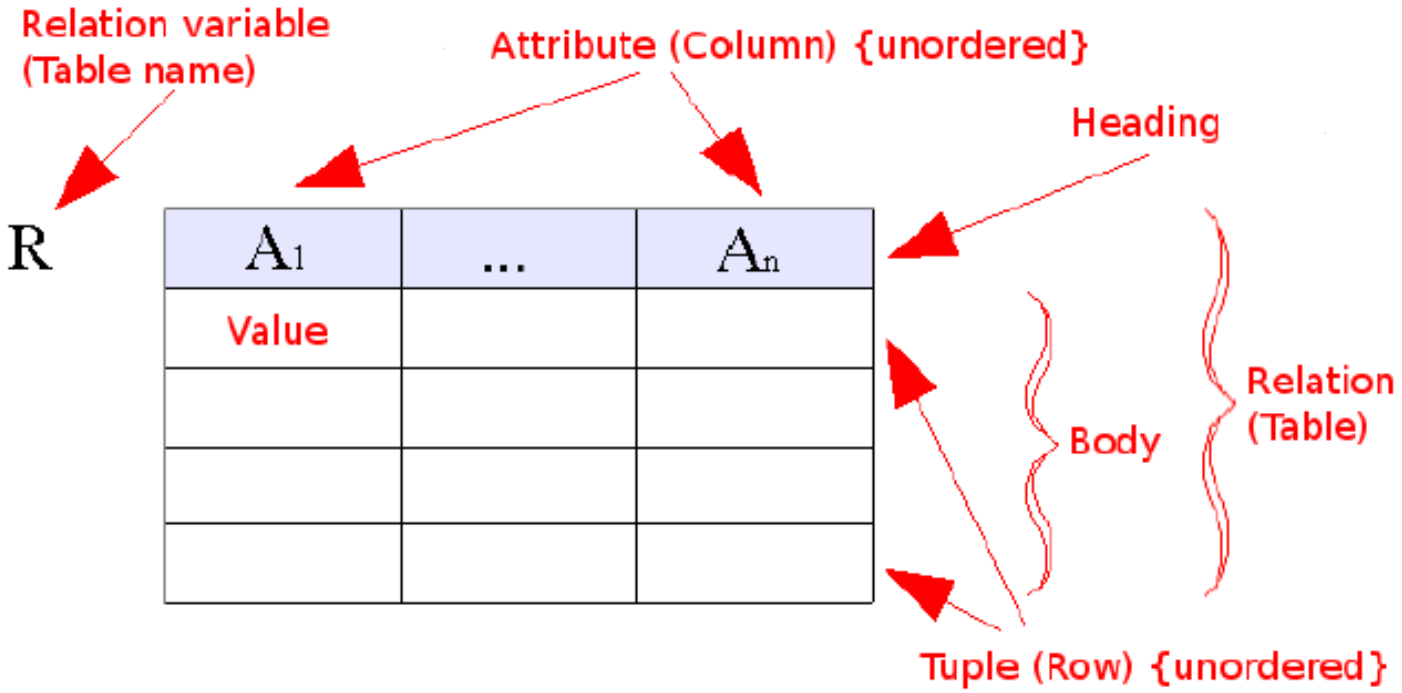
Relational Model Concepts

- **Relation Instance**
  - Relation instance is a finite set of tuples in the RDBMS system.
  - Relation instances never have duplicate tuples.
- **Relation Key**
  - Every row has one, two, or multiple attributes, which is called relation key.
- **Attribute Domain**
  - Every attribute has some pre-defined value and scope which is known as attribute domain.

Relational Model Concepts



Relational Model Concepts



Relational Integrity Constraints

- Relational Integrity Constraints is referred to conditions which must be present for a valid relation.
- These integrity constraints are derived from the rules in the mini-world that the database represents.
- There are many types of integrity constraints.
- Constraints on the RDBMS is mostly divided into three main categories:
  - Domain Constraints
  - Key Constraints
  - Referential Integrity Constraints

Domain Constraints

- Domain constraints can be violated if an attribute value is not appearing in the correspondence domain or it is not of the appropriate data type.
- Domain constraints specify that within each tuple, and the value of each attribute must be unique.
- This is specified as data types which include standard data types integers, real numbers, characters, Booleans, variable length strings, etc.

Domain Constraints

- Example:

  Create DOMAIN CustomerName

  CHECK (value not NULL)

- The example shown demonstrates creating a domain constraint such that *CustomerName* is not NULL.

Key Constraints

- An attribute that can uniquely identify a tuple in a relation is called the key of the table.
- The value of the attribute for different tuples in the relation has to be unique.

In the given table, CustomerID is a key attribute of Customer Table.

It is most likely to have a single key for one customer.

i.e. CustomerID=1 is only for the CustomerName="Google".

CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Inactive

Referential Integrity Constraints

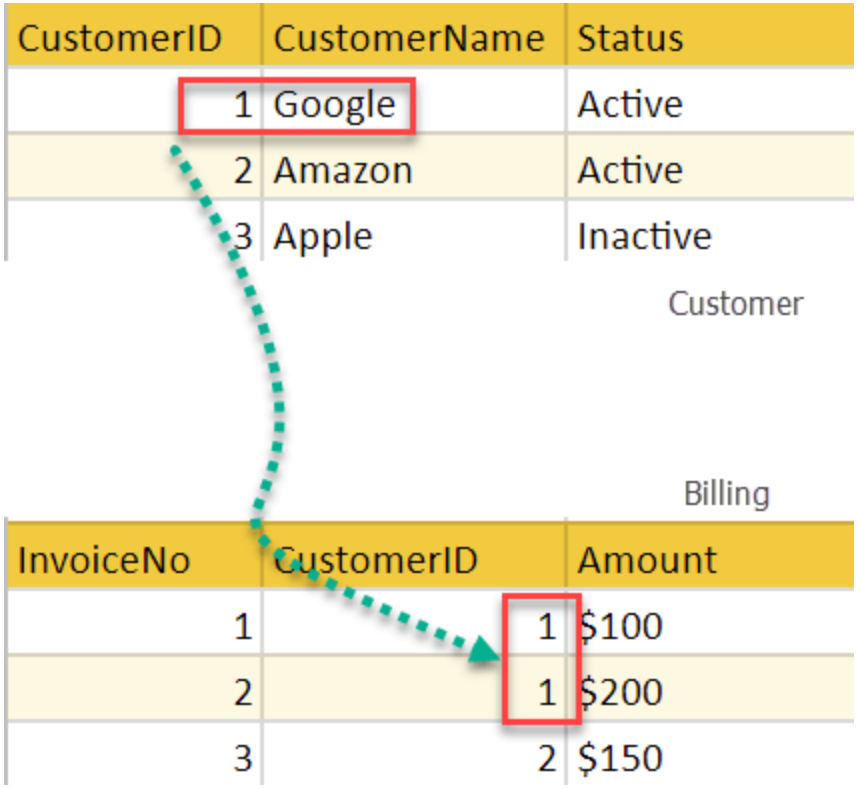
- Referential integrity constraints is based on the concept of Foreign keys.
- A foreign key is an important attribute of a relation which should be referred to in other relationships.
- Referential integrity constraint state happens where relation refers to a key attribute of a different or same relation.
- However, that key element must exist in the table.

Referential Integrity Constraints

Here we have 2 relations, Customer and Billing

Tuple for CustomerID=1 is referenced twice in the relation Billing.

So we know CustomerName=Google has billing amount of $300



CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Inactive

Customer

Billing

InvoiceNo	CustomerID	Amount
1	1	$100
2	1	$200
3	2	$150

Advantages of Relational Model

- **Simplicity**
  - A relational data model is simpler than the hierarchical and network model
- **Structural Independence**
  - The relational database is only concerned with data and not with a structure.
  - This can improve the performance of the model.
- **Easy to use**
  - The relational model is easy as tables consisting of rows and columns is quite natural and simple to understand.

Advantages of Relational Model

- **Query Capability**
  - It makes possible for a high-level query language like SQL to avoid complex database navigation.
- **Data Independence**
  - The structure of a database can be changed without having to change any application.
- **Scalable**
  - Regarding a number of records, or rows, and the number of fields, a database should be enlarged to enhance its usability.

Disadvantages of Relational Model

- Few relational databases have limits on field lengths which can't be exceeded.
- Relational databases can sometimes become complex as the amount of data grows, and the relations between pieces of data become more complicated.
- Complex relational database systems may lead to isolated databases where the information cannot be shared from one system to another.

Codd's Rules for RDBMS

- Dr. Edgar Frank Codd *(August 19, 1923 – April 18, 2003)* was a computer scientist while working for IBM he invented the relational model for database management *(theoretical basis for relational database)*.
- Codd proposed thirteen rules *(numbered zero to twelve)* and said that if a Database Management System (DBMS) meets these rules, it can be called as a Relational Database Management System (RDBMS).

Codd's Rules for RDBMS

- **Rule Zero**
  - The system must qualify as relational, as a database, and as a management system. For a system to qualify as a RDBMS, that system must use its relational facilities (exclusively) to manage the database.
  - The other 12 rules derive from this rule.
- **Rule 1 – the information rule**
  - All information in the database is to be represented in one and only one way, namely by values in column positions within rows of tables.

Codd's Rules for RDBMS

- **Rule 2 – the guaranteed access rule**
  - All data must be accessible.
  - This rule is essentially a restatement of the fundamental requirement for primary keys.
  - It says that every individual scalar value in the database must be logically addressable by specifying the name of the containing table, the name of the containing column and the primary key value of the containing row.

Codd's Rules for RDBMS

- **Rule 3 – systematic treatment of null values**
  - The DBMS must allow each field to remain null (or empty).
  - Specifically, it must support a representation of "missing information and inapplicable information" that is systematic, distinct from all regular values. (for example, "distinct from zero or any other number", in the case of numeric values), and independent of data type.
  - It is also implied that such representations must be manipulated by the DBMS in a systematic way.

Codd's Rules for RDBMS

- **Rule 4 – active online catalog based on the relational model**
  - The system must support an online, inline, relational catalog that is accessible to authorized users by means of their regular query language.
  - That is, users must be able to access the database's structure (catalog) using the same query language that they use to access the database's data.
- **Rule 5 – the comprehensive data sub language rule**
  - The system must support at least one relational language that
    - Has a linear syntax
    - Can be used both interactively and within application programs
    - Supports data definition operations (including view definitions), data manipulation operations (update as well as retrieval), security and integrity constraints, and transaction management operations (begin, commit, and rollback).

Codd's Rules for RDBMS

- **Rule 6 – the view updating rule**
  - All views those can be updated theoretically, must be updated by the system.
- **Rule 7 – high level insert, update, and delete**
  - The system must support set-at-a-time insert, update, and delete operators.
  - This means that data can be retrieved from a relational database in sets constructed of data from multiple rows and/or multiple tables.
  - This rule states that insert, update, and delete operations should be supported for any retrievable set rather than just for a single row in a single table.

Codd's Rules for RDBMS

- **Rule 8 – physical data independence**
  - Changes to the physical level (how the data is stored, whether in arrays or linked lists, etc.) must not require a change to an application based on the structure.
- **Rule 9 – logical data independence**
  - Changes to the logical level (tables, columns, rows, and so on) must not require a change to an application based on the structure.
  - Logical data independence is more difficult to achieve than physical data independence.
- **Rule 10 – integrity independence**
  - Integrity constraints must be specified separately from application programs and stored in the catalog.
  - It must be possible to change such constraints as and when appropriate without unnecessarily affecting existing applications.

Codd's Rules for RDBMS

- **Rule 11 – distribution independence**
  - The distribution of portions of the database to various locations should be invisible to users of the database.
  - Existing applications should continue to operate successfully:
    - When a distributed version of the DBMS is first introduced.
    - When an existing distributed data are redistributed around the system.
- **Rule 12 – the non-subversion rule**
  - If the system provides a low-level (record-at-a-time) interface, then that interface cannot be used to subvert the system.
  - For example, bypassing a relational security or integrity constraint.

END OF UNIT TWO

Syllabus

- UNIT 3: Retrieving Data
  - SQL Basics
    - Name: Table Names; Column Names
    - Data Types
    - Constants
  - Simple Queries
    - The SELECT Statement: The SELECT Clause; The FROM Clause
    - Multi-table Queries (Joins)
    - Duplicate Rows
    - Row Selection
    - Search Conditions
    - The Comparison Test (=, <, >, <=, >=)
    - The Range Test (BETWEEN)
    - The Set Membership Test (IN)
    - The Pattern Matching Test (LIKE)
    - The Null Value Test (IS NULL)
    - Compound Search Conditions (AND, OR, and NOT)
    - Sorting Query Results (ORDER BY Clause)

UNIT 3: Retrieving Data

SQL – Data Types

- SQL Data Type is an attribute that specifies the type of data of any object.
- Each column, variable and expression has a related data type in SQL.
- You can use these data types while creating your tables.
- You can choose a data type for a column based on your requirement.
- SQL Server offers six categories of data types for your use which are listed below:
  - Exact Numeric Data Types
  - Approximate Numeric Data Types
  - Data and Time Data Types
  - Character Strings Data Types
  - Unicode Character Strings Data Types
  - Binary Data Types
  - Miscellaneous Data Types

Exact Numeric Data Types

DATA TYPE	FROM	TO
bigint	-9,223,372,036,854,775,808	9,223,372,036,854,775,807
int	-2,147,483,648	2,147,483,647
smallint	-32,768	32,767
tinyint	0	255
bit	0	1
decimal	-10^38 +1	10^38 -1
numeric	-10^38 +1	10^38 -1
money	-922,337,203,685,477.5808	+922,337,203,685,477.5807
smallmoney	-214,748.3648	+214,748.3647

Approximate Numeric Data Types

DATA TYPE	FROM	TO
float	-1.79E + 308	1.79E + 308
real	-3.40E + 38	3.40E + 38

Date and Time Data Types

DATA TYPE	FROM	TO
datetime	Jan 1, 1753	Dec 31, 9999
smalldatetime	Jan 1, 1900	Jun 6, 2079
date	Stores a date like June 30, 1991	
time	Stores a time of day like 12:30 P.M.	

NOTE – Here, *datetime* has 3.33 milliseconds accuracy whereas *smalldatetime* has 1 minute accuracy.

Character Strings Data Types

Sr. No.	DATA TYPE & Description
1	**char** Maximum length of 8,000 characters (Fixed length non-Unicode characters).
2	**varchar** Maximum length of 8,000 characters (Variable-length non-Unicode data).
3	**varchar(max)** Maximum length of 2E+31 characters, variable-length non-Unicode data.
4	**text** Variable-length non-Unicode data with a maximum length of 2,147,483,647 characters.

Unicode Character Strings Data Types

Sr. No.	DATA TYPE & Description
1	**nchar** Maximum length of 4,000 characters (Fixed length Unicode).
2	**nvarchar** Maximum length of 4,000 characters (Variable-length Unicode).
3	**nvarchar(max)** Maximum length of 2E+31 characters (Variable-length Unicode)
4	**ntext** Maximum length of 1,073,741,823 characters (Variable length Unicode)

Binary Data Types

Sr. No.	DATA TYPE & Description
1	**binary** Maximum length of 8,000 bytes (Fixed-length binary data).
2	**varbinary** Maximum length of 8,000 bytes(Variable length binary data).
3	**varbinary(max)** Maximum length of 2E+31 bytes (Variable length Binary data)
4	**image** Maximum length of 2,147,483,647 bytes (Variable length Binary Data)

Miscellaneous Data Types

Sr. No.	DATA TYPE & Description
1	**sql_variant** Stores values of various SQL Server-supported data types, except text, ntext, and timestamp.
2	**timestamp** Stores a database-wide unique number that gets updated every time a row gets updated.
3	**uniqueidentifier** Stores a globally unique identifier (GUID).
4	**xml** Stores XML data. You can store xml instances in a column or a variable.
5	**cursor** Reference to a cursor object.
6	**table** Stores a result set for later processing.

CREATE TABLE

- Creating a basic table involves naming the table and defining its columns and each column's data type
- The SQL **CREATE TABLE** statement is used to create a new table.
- Syntax:

```
CREATE TABLE table_name(
        column1 datatype,
        column2 datatype,
        …
        columnN datatype,
        PRIMARY KEY (one or more columns)
);
```

CREATE TABLE

- **CREATE TABLE** is the keyword telling the database system what you want to do.
- In this case, you want to create a new table.
- The unique name or identifier for the table follows the **CREATETABLE** statement.
- Then in brackets comes the list defining each column in the table and what sort of data type it is.

CREATE TABLE– Example

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

- The following code block is an example, which creates a **CUSTOMERS** table with an **ID** as a primary key and **NOT NULL** are the constraints showing that these fields cannot be NULL while creating records in this table.
- You can verify if your table has been created successfully by looking at the message displayed by the SQL server, otherwise you can use the **DESC** command.

CREATE TABLE– Example

```
Name     Null?    Type
-------  -------- ------------
ID       NOT NULL NUMBER(38)
NAME     NOT NULL VARCHAR2(20)
AGE      NOT NULL NUMBER(38)
ADDRESS           CHAR(25)
SALARY            NUMBER(18,2)
```

- Now, you have CUSTOMERS table available in your database which you can use to store the required information related to customers.

DROP or DELETE Table

- The SQL **DROP TABLE** statement is used to remove a table definition and all the data, indexes, triggers, constraints, and permission specifications for that table.
- NOTE – you should be very careful while using this command because once a table is deleted then all the information available in that table will also be lost forever.
- Syntax:

  DROP TABLE table_name;

DROP or DELETE Table – Example

- Let us first verify the CUSTOMERS table and then we will delete it from the database.

  DESC CUSTOMERS;

```
Name     Null?    Type
-------  -------- ------------
ID       NOT NULL NUMBER(38)
NAME     NOT NULL VARCHAR2(20)
AGE      NOT NULL NUMBER(38)
ADDRESS           CHAR(25)
SALARY            NUMBER(18,2)
```

- This means that the CUSTOMERS table is available in the database, so let us now drop it as shown below.

  DROP TABLE CUSTOMERS;

- Now, if you would try the DESC command, then you will get the following error:

  ERROR:

  ORA-04043: object CUSTOMERS does not exist

INSERT Query

- The SQL **INSERT INTO** statement is used to add new rows of data to a table in the database.
- Syntax:

  INSERT INTO TABLE_NAME (column1, column2, …, columnN) VALUES (value1, value2, …, valueN);

  Here, column1, column2, …, columnN are the names of the columns in the table into which you want to insert the data.
- You may not need to specify the column(s) name in the SQL query if you are adding values for all the columns of the table. But make sure the order of the values is in the same order as the columns in the table.
- The SQL **INSERT INTO** syntax will be as follows:

  INSERT INTO TABLE_NAME VALUES (value1, value2, …, valueN);

INSERT Query – Example

- The following statements would create seven records in the **CUSTOMERS** table:

```
INSERT INTO CUSTOMERS (ID, NAME, AGE, ADDRESS, SALARY)
VALUES (1, 'Anil', 32, 'Kathmandu', 2000.00);
INSERT INTO CUSTOMERS (ID, NAME, AGE, ADDRESS, SALARY)
VALUES (2, 'Sandace', 25, 'Pokhara', 1500.00);
INSERT INTO CUSTOMERS (ID, NAME, AGE, ADDRESS, SALARY)
VALUES (3, 'Prashant', 23, 'Chitwan', 2000.00);
INSERT INTO CUSTOMERS (ID, NAME, AGE, ADDRESS, SALARY)
VALUES (4, 'Santosh', 25, 'Biratnagar', 6500.00);
INSERT INTO CUSTOMERS (ID, NAME, AGE, ADDRESS, SALARY)
VALUES (5, 'Shreeram', 28, 'Lumbini', 8500.00);
INSERT INTO CUSTOMERS (ID, NAME, AGE, ADDRESS, SALARY)
VALUES (6, 'Krishna', 22, 'Lamjung', 4500.00);
INSERT INTO CUSTOMERS (ID, NAME, AGE, ADDRESS, SALARY)
VALUES (7, 'Sonam', 22, 'Bhojpur', 10000.00);
```

INSERT Query – Example

- The alternative way to create same seven records in **CUSTOMERS** table:

```
INSERT INTO CUSTOMERS VALUES (1, 'Anil', 32, 'Kathmandu', 2000.00);
INSERT INTO CUSTOMERS VALUES (2, 'Sandace', 25, 'Pokhara', 1500.00);
INSERT INTO CUSTOMERS VALUES (3, 'Prashant', 23, 'Chitwan', 2000.00);
INSERT INTO CUSTOMERS VALUES (4, 'Santosh', 25, 'Biratnagar', 6500.00);
INSERT INTO CUSTOMERS VALUES (5, 'Shreeram', 28, 'Lumbini', 8500.00);
INSERT INTO CUSTOMERS VALUES (6, 'Krishna', 22, 'Lamjung', 4500.00);
INSERT INTO CUSTOMERS VALUES (7, 'Sonam', 22, 'Bhojpur', 10000.00);
```

INSERT Query – Example

- All the statements from earlier slides would produce the following records in the **CUSTOMERS** table as shown below:

ID	NAME	AGE	ADDRESS	SALARY
1	Anil	32	Kathmandu	2000
2	Sandace	25	Pokhara	1500
3	Prashant	23	Chitwan	2000
4	Santosh	25	Biratnagar	6500
5	Shreeram	28	Lumbini	8500
6	Krishna	22	Lamjung	4500
7	Sonam	22	Bhojpur	10000

SELECT Query

- The SQL **SELECT** statement is used to fetch the data from a database table which returns this data in the form of a result table.
- These result tables are called result-sets.
- Syntax:

  SELECT column1, column2, ..., columnN FROM table_name;

  Here, column1, column2, ... are the fields of a table whose values you want to fetch.
- If you want to fetch all the fields available in the field, then you can use the following syntax:

  *SELECT * FROM table_name;*

SELECT Query – Example

- Consider the CUSTOMERS table having the following records:
- The following code is an example, which would fetch the ID, Name, and Salary fields of the customers available in CUSTOMERS table.

  SELECT ID, NAME, SALARY FROM CUSTOMERS;
- This would produce the following result:

ID	NAME	AGE	ADDRESS	SALARY
1	Anil	32	Kathmandu	2000
2	Sandace	25	Pokhara	1500
3	Prashant	23	Chitwan	2000
4	Santosh	25	Biratnagar	6500
5	Shreeram	28	Lumbini	8500
6	Krishna	22	Lamjung	4500
7	Sonam	22	Bhojpur	10000

ID	NAME	SALARY
1	Anil	2000
2	Sandace	1500
3	Prashant	2000
4	Santosh	6500
5	Shreeram	8500
6	Krishna	4500
7	Sonam	10000

SELECT Query - EXAMPLE

- If you want to fetch all the fields of the CUSTOMERS table, then you should use the following query:

  *SELECT * FROM CUSTOMERS;*

- This would produce the result as shown below:

ID	NAME	AGE	ADDRESS	SALARY
1	Anil	32	Kathmandu	2000
2	Sandace	25	Pokhara	1500
3	Prashant	23	Chitwan	2000
4	Santosh	25	Biratnagar	6500
5	Shreeram	28	Lumbini	8500
6	Krishna	22	Lamjung	4500
7	Sonam	22	Bhojpur	10000

WHERE Clause

- The SQL **WHERE** clause is used to specify a condition while fetching the data from a single table or by joining with multiple tables.
- If the given condition is satisfied, then only it returns a specific value from the table.
- You should use the **WHERE** clause to filter the records and fetching only the necessary records.
- The **WHERE** clause is not only used in the *SELECT* statement, but it is also used in the *UPDATE*, *DELETE* statement, etc.
- Syntax:

  SELECT column1, column2, ..., columnN
  FROM table_name
  WHERE [condition];

- You can specify a condition using the comparison or logical operators like >, <, =, LIKE, NOT, etc.

WHERE Clause – Example

- Consider the **CUSTOMERS** table having the following records:

ID	NAME	AGE	ADDRESS	SALARY
1	Anil	32	Kathmandu	2000
2	Sandace	25	Pokhara	1500
3	Prashant	23	Chitwan	2000
4	Santosh	25	Biratnagar	6500
5	Shreeram	28	Lumbini	8500
6	Krishna	22	Lamjung	4500
7	Sonam	22	Bhojpur	10000

- The following code is an example which would fetch the ID, Name, and Salary fields from the CUSTOMERS table, where the salary is greater than 2000:

```
SELECT ID, NAME, SALARY
FROM CUSTOMERS
WHERE SALARY>2000;
```

- This would produce the following result:

ID	NAME	SALARY
4	Santosh	6500
5	Shreeram	8500
6	Krishna	4500
7	Sonam	10000

WHERE Clause – Example

- The following query is an example, which would fetch the ID, Name, and Salary fields from the CUSTOMERS table for a customer with the name Shreeram.

```
SELECT ID, NAME, SALARY
FROM CUSTOMERS
WHERE NAME='Shreeram';
```

- Here, it is important to note that all the strings should be given inside single quotes (‘ ’). Whereas, numeric values should be given without any quote as in the earlier example.

ID	NAME	AGE	ADDRESS	SALARY
1	Anil	32	Kathmandu	2000
2	Sandace	25	Pokhara	1500
3	Prashant	23	Chitwan	2000
4	Santosh	25	Biratnagar	6500
5	Shreeram	28	Lumbini	8500
6	Krishna	22	Lamjung	4500
7	Sonam	22	Bhojpur	10000

ID	NAME	SALARY
5	Shreeram	8500

AND & OR Conjunctive Operators

- The SQL **AND** & **OR** operators are used to combine multiple conditions to narrow data in an SQL statement.
- These two operators are called as the **conjunctive operators**.
- These operators provide a means to make multiple comparisons with different operators in the same SQL statement.
- The **AND** operator allows the existence of multiple conditions in an SQL statement's **WHERE** clause.
- The **OR** operator is used to combine multiple conditions in an SQL statement's **WHERE** clause.

The AND Operator

- Syntax:

  SELECT column1, column2, …, columnN

  FROM table_name

  WHERE [condition1] AND [condition2] … AND [conditionN];

- You can combine N number of conditions using the AND operator.
- For an action to be taken by the SQL statement, whether it be a transaction or a query, all conditions separated by the AND must be TRUE.

The AND Operator – EXAMPLE

- Consider the CUSTOMERS table having the following records.
- Following is an example, which would fetch the ID, Name, & Salary fields from the CUSTOMERS table, where the salary is greater than 2000 and the age is less than 25 years:

```
SELECT ID, NAME, SALARY
FROM CUSTOMERS
WHERE SALARY>2000 AND
AGE<25;
```

- This would produce the following result:

ID	NAME	AGE	ADDRESS	SALARY
1	Anil	32	Kathmandu	2000
2	Sandace	25	Pokhara	1500
3	Prashant	23	Chitwan	2000
4	Santosh	25	Biratnagar	6500
5	Shreeram	28	Lumbini	8500
6	Krishna	22	Lamjung	4500
7	Sonam	22	Bhojpur	10000

```
 ID NAME                 SALARY
--- -------------------- ----------
  6 Krishna                4500
  7 Sonam                 10000
```

The OR Operator

- Syntax:

  SELECT column1, column2, …, columnN

  FROM table_name

  WHERE [condition1] OR [condition2] … OR [conditionN];

- You can combine N number of conditions using the OR operator.
- For an action to be taken by the SQL statement, whether it be a transaction or a query, the only any ONE of the conditions separated by the OR must be TRUE.

The OR Operator – EXAMPLE

- Consider the CUSTOMERS table having the following records.

ID	NAME	AGE	ADDRESS	SALARY
1	Anil	32	Kathmandu	2000
2	Sandace	25	Pokhara	1500
3	Prashant	23	Chitwan	2000
4	Santosh	25	Biratnagar	6500
5	Shreeram	28	Lumbini	8500
6	Krishna	22	Lamjung	4500
7	Sonam	22	Bhojpur	10000

- Following is an example, which would fetch the ID, Name, & Salary fields from the CUSTOMERS table, where the salary is greater than 2000 or the age is less than 25 years:

```
SELECT ID, NAME, SALARY
FROM CUSTOMERS
WHERE SALARY>2000 OR AGE<25;
```

- This would produce the following result:

```
 ID NAME                 SALARY
--- -------------------- ----------
  3 Prashant                 2000
  4 Santosh                  6500
  5 Shreeram                 8500
  6 Krishna                  4500
  7 Sonam                   10000
```

UPDATE Query

- The SQL **UPDATE** query is used to modify the existing records in a table.
- You can use the WHERE clause with the UPDATE query to update the selected rows, otherwise all the rows would be affected.
- Syntax:

  UPDATE table_name

  SET column1=value1, column2=value2, …, columnN=valueN

  WHERE [condition];

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

UPDATE Query – EXAMPLE

- Consider the CUSTOMERS table having the following records.
- The following query will update the ADDRESS for a customer whose ID number is 6 in the table

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

- Now, the CUSTOMERS table would have the following records.

ID	NAME	AGE	ADDRESS	SALARY
1	Anil	32	Kathmandu	2000
2	Sandace	25	Pokhara	1500
3	Prashant	23	Chitwan	2000
4	Santosh	25	Biratnagar	6500
5	Shreeram	28	Lumbini	8500
6	Krishna	22	Lamjung	4500
7	Sonam	22	Bhojpur	10000

ID	NAME	AGE	ADDRESS	SALARY
1	Anil	32	Kathmandu	2000
2	Sandace	25	Pokhara	1500
3	Prashant	23	Chitwan	2000
4	Santosh	25	Biratnagar	6500
5	Shreeram	28	Lumbini	8500
6	Krishna	22	Dang	4500
7	Sonam	22	Bhojpur	10000

UPDATE Query – EXAMPLE

- If you want to modify all the ADDRESS and the SALARY column values in the CUSTOMERS table, you do not need to use the WHERE clause as the UPDATE query would be enough as shown in the following query below:
- The following query will update the ADDRESS for a customer whose ID number is 6 in the table

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

- Now, the CUSTOMERS table would have the following records.

ID	NAME	AGE	ADDRESS	SALARY
1	Anil	32	Dang	1000
2	Sandace	25	Dang	1000
3	Prashant	23	Dang	1000
4	Santosh	25	Dang	1000
5	Shreeram	28	Dang	1000
6	Krishna	22	Dang	1000
7	Sonam	22	Dang	1000

DELETE Query

- The SQL **DELETE** query is used to delete the existing records from a table.
- You can use the WHERE clause with a DELETE query to delete the selected rows, otherwise all the records would be deleted.
- Syntax:

  DELETE FROM table_name

  WHERE [condition];

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

DELETE Query – EXAMPLE

- Consider the CUSTOMERS table having the following records.
- The following code has a query, which will DELETE a customer, whose ID is 6

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

- Now, the CUSTOMERS table would have the following records.

ID	NAME	AGE	ADDRESS	SALARY
1	Anil	32	Kathmandu	2000
2	Sandace	25	Pokhara	1500
3	Prashant	23	Chitwan	2000
4	Santosh	25	Biratnagar	6500
5	Shreeram	28	Lumbini	8500
6	Krishna	22	Lamjung	4500
7	Sonam	22	Bhojpur	10000

ID	NAME	AGE	ADDRESS	SALARY
1	Anil	32	Kathmandu	2000
2	Sandace	25	Pokhara	1500
3	Prashant	23	Chitwan	2000
4	Santosh	25	Biratnagar	6500
5	Shreeram	28	Lumbini	8500
7	Sonam	22	Bhojpur	10000

DELETE Query – EXAMPLE

- Consider the CUSTOMERS table having the following records.
- Now if you want to delete all the records from the CUSTOMERS table, you do not need to use the WHERE clause and the DELETE query would be as follows:

  DELETE FROM CUSTOMERS
- Now, the CUSTOMERS table would have no any record.

ID	NAME	AGE	ADDRESS	SALARY
1	Anil	32	Kathmandu	2000
2	Sandace	25	Pokhara	1500
3	Prashant	23	Chitwan	2000
4	Santosh	25	Biratnagar	6500
5	Shreeram	28	Lumbini	8500
6	Krishna	22	Lamjung	4500
7	Sonam	22	Bhojpur	10000

LIKE Clause

- The SQL **LIKE** clause is used to compare a value to similar values using wildcard operators.
- There are two wildcards used in conjunction with the LIKE operator.
  - *The percent sign (%)*
  - *The underscore (_)*
- The **percent (%)** sign represents zero, one or multiple characters.
- The **underscore (_)** sign represents a single number or character.
- These symbols can be used in combinations.

LIKE Clause

- Syntax:

  SELECT FROM table_nameWHERE column LIKE 'XXXX%'

  or

  SELECT FROM table_nameWHERE column LIKE '%XXXX%'

  or

  SELECT FROM table_nameWHERE column LIKE 'XXXX_'

  or

  SELECT FROM table_nameWHERE column LIKE '_XXXX'

  or

  SELECT FROM table_nameWHERE column LIKE '_XXXX_'

- You can combine N number of conditions using AND or OR operators.
- Here, *XXXX* could be any numeric or string value.

LIKE Clause – Example

- The following table has a few examples showing the WHERE part having different LIKE clause with '%' and '_' operators:

Sr. No.	Statement & Description
1	WHERE SALARY LIKE '200%' - Finds any values that start with 200.
2	WHERE SALARY LIKE '%200%' - Finds any values that have 200 in any position.
3	WHERE SALARY LIKE '_00%' - Finds any values that have 00 in the second and third positions.
4	WHERE SALARY LIKE '2_%_%' - Finds any values that start with 2 and are at least 3 characters in length.
5	WHERE SALARY LIKE '%2' - Finds any values that end with 2.
6	WHERE SALARY LIKE '_2%3' - Finds any values that have a 2 in the second position and end with a 3.
7	WHERE SALARY LIKE '2_ _ _ 3' - Finds any values in a five digit number that starts with 2 and end with 3.

LIKE Clause – EXAMPLE

- Consider the CUSTOMERS table having the following records.

ID	NAME	AGE	ADDRESS	SALARY
1	Anil	32	Kathmandu	2000
2	Sandace	25	Pokhara	1500
3	Prashant	23	Chitwan	2000
4	Santosh	25	Biratnagar	6500
5	Shreeram	28	Lumbini	8500
6	Krishna	22	Lamjung	4500
7	Sonam	22	Bhojpur	10000

- Following is an example, which would display all the records from the CUSTOMERS table, where the SALARY starts with 200.

```
SELECT * FROM CUSTOMERS
WHERE SALARY LIKE '200%';
```

- Now, the CUSTOMERS table would have the following records.

ID	NAME	AGE	ADDRESS	SALARY
1	Anil	32	Kathmandu	2000
3	Prashant	23	Chitwan	2000

ORDER BY Clause

- The SQL **ORDER BY** clause is used to sort the data in ascending or descending order, based on one or more columns.
- Some databases sort the query results in an ascending order by default.
- Syntax:

  SELECT column-list

  FROM table_name

  WHERE [condition]

  ORDER BY [column1, column2, …, columnN] [ASC | DESC];

- You can use more than one column in the ORDER BY clause.
- Make sure whatever column you are using to sort that column should be in the column-list.

ORDER BY Clause – EXAMPLE

- Consider the CUSTOMERS table having the following records.
- The following code block has an example, which would sort the result in an ascending order by the NAME and the SALARY:

```
SELECT * FROM CUSTOMERS
ORDER BY NAME;
```

- The following code block has an example, which would sort the result in the descending order by NAME:

```
SELECT * FROM CUSTOMERS
ORDER BY NAME DESC;
```

ID	NAME	AGE	ADDRESS	SALARY
1	Anil	32	Kathmandu	2000
2	Sandace	25	Pokhara	1500
3	Prashant	23	Chitwan	2000
4	Santosh	25	Biratnagar	6500
5	Shreeram	28	Lumbini	8500
6	Krishna	22	Lamjung	4500
7	Sonam	22	Bhojpur	10000

ID	NAME	AGE	ADDRESS	SALARY
1	Anil	32	Kathmandu	2000
6	Krishna	22	Lamjung	4500
3	Prashant	23	Chitwan	2000
2	Sandace	25	Pokhara	1500
4	Santosh	25	Biratnagar	6500
5	Shreeram	28	Lumbini	8500
7	Sonam	22	Bhojpur	10000

ID	NAME	AGE	ADDRESS	SALARY
7	Sonam	22	Bhojpur	10000
5	Shreeram	28	Lumbini	8500
4	Santosh	25	Biratnagar	6500
2	Sandace	25	Pokhara	1500
3	Prashant	23	Chitwan	2000
6	Krishna	22	Lamjung	4500
1	Anil	32	Kathmandu	2000

GROUP BY Clause

- The SQL **GROUP BY** clause is used in collaboration with the SELECT statement to arrange identical data into groups.
- This GROUP BY clause follows the WHERE clause in a SELECT statement and precedes the ORDER BY clause.
- Syntax:

```
SELECT column-list
FROM table_name
WHERE [condition]
GROUP BY column1, column2
ORDER BY [column1, column2, ..., columnN] [ASC | DESC];
```

- You can use more than one column in the GROUP BY clause.

GROUP BY Clause – EXAMPLE

- Consider the CUSTOMERS table having the following records.
- If you want to know the total amount of the salary on each customer, then the GROUP BY query would be as follows:

SELECT NAME, SUM(SALARY)
FROM CUSTOMERS
GROUP BY NAME;

- This would produce the following result.

ID	NAME	AGE	ADDRESS	SALARY
1	Anil	32	Kathmandu	2000
2	Sandace	25	Pokhara	1500
3	Prashant	23	Chitwan	2000
4	Santosh	25	Biratnagar	6500
5	Shreeram	28	Lumbini	8500
6	Krishna	22	Lamjung	4500
7	Sonam	22	Bhojpur	10000

NAME	SUM(SALARY)
Anil	2000
Santosh	6500
Sandace	1500
Prashant	2000
Shreeram	8500
Krishna	4500
Sonam	10000

GROUP BY Clause – EXAMPLE

- Now, let us look at a table where the CUSTOMER table has the following records with duplicate names.
- Now again, if you want to know the total amount of SALARY on each customer, then the GROUP BY query would be as follows:

  SELECT NAME, SUM(SALARY) FROM CUSTOMER

  GROUP BY NAME;

- This would produce the following result.

ID	NAME	AGE	ADDRESS	SALARY
1	Anil	32	Kathmandu	2000
2	Anil	25	Pokhara	1500
3	Prashant	23	Chitwan	2000
4	Prashant	25	Biratnagar	6500
5	Shreeram	28	Lumbini	8500
6	Krishna	22	Lamjung	4500
7	Sonam	22	Bhojpur	10000

NAME	SUM(SALARY)
Anil	3500
Prashant	8500
Shreeram	8500
Krishna	4500
Sonam	10000

DISTINCT Keyword

- The SQL **DISTINCT** keyword is used in conjunction with the SELECT statement to eliminate all the duplicate records and fetching only unique records.
- There may be a situation when you have multiple duplicate records in a table.
- While fetching such records, it makes more sense to fetch only those unique records instead of fetching duplicate records.
- Syntax:

```
SELECT DISTINCT column-list
FROM table_name
WHERE [condition];
```

DISTINCT – EXAMPLE

- Consider the CUSTOMERS table having the following records.
- First let us see how the following SELECT query returns the duplicate salary records:

```
SELECT SALARY
FROM CUSTOMERS
ORDER BY SALARY;
```

- Now, let us use the DISTINCT keyword with the above SELECT query:

```
SELECT DISTINCT SALARY
FROM CUSTOMERS
ORDER BY SALARY;
```

ID	NAME	AGE	ADDRESS	SALARY
1	Anil	32	Kathmandu	2000
2	Anil	25	Pokhara	1500
3	Prashant	23	Chitwan	2000
4	Prashant	25	Biratnagar	6500
5	Shreeram	28	Lumbini	8500
6	Krishna	22	Lamjung	4500
7	Sonam	22	Bhojpur	10000

SALARY
1500
2000
2000
4500
6500
8500
10000

SALARY
1500
2000
4500
6500
8500
10000

Constraints

- Constraints are the rules enforced on the data columns of a table.
- These are used to limit the type of data that can go into a table.
- This ensures the accuracy and reliability of the data in the database.
- Constraints could be either on a column level or a table level.
- The column level constraints are applied only to one column, whereas the table level constraints are applied to the whole table.

Constraints

- Following are some of the most commonly used constraints available in SQL.

Constraints	Description
NOT NULL	Ensures that a column cannot have NULL value.
DEFAULT	Provides a default value for a column when none is specified.
UNIQUE	Ensures that all values in a column are different.
PRIMARY	Uniquely identifies each row/record in a database table.
FOREIGN	Uniquely identifies a row/record in any of the given database table.
CHECK	Ensures that all the values in a column satisfies certain conditions.
INDEX	Used to create and retrieve data from the database very quickly.

- Constraints can be specified when a table is created with the CREATE TABLE statement or you can use the ALTER TABLE statement to create constraints even after the table is created.

Dropping Constraints

- Any constraint that you have defined can be dropped using the ALTER TABLE command with the DROP CONSTRAINT option.
- For example, to drop the primary key constraint in the EMPLOYEES table, you can use the following command:

  ALTER TABLE EMPLOYEES DROP CONSTRAINT EMPLOYEES_PK;
- Some implementations may provide shortcuts for dropping certain constraints.
- For example, to drop the primary key constraint for a table in Oracle, you can use the following command:

  ALTER TABLE EMPLOYEES DROP PRIMARY KEY;
- Some implementations allow you to disable constraints instead of permanently dropping it from the database.
- You may want to temporarily disable the constraint and then enable it later.

Integrity Constraints

- Integrity constraints are used to ensure accuracy and consistency of the data in a relational database.
- Data integrity is handled in a relational database through the concept of referential integrity.
- There are many types of integrity constraints that play a role in **Referential Integrity (RI)**.
- These constraints include Primary Key, Foreign Key, Unique Constraints and other constraints which are mentioned in earlier slide.

Using Joins

- The SQL **Joins** clause is used to combine records from two or more tables in a database.
- A JOIN is a means for combining fields from two tables by using values common to each.
- Let us first create another table ORDERS using the following query:

```
CREATE TABLE ORDERS(
OID INT NOT NULL,
DATE_OF_ORDER DATE NOT NULL,
CUSTOMERS_ID INT NOT NULL,
AMOUNT INT NOT NULL,
PRIMARY KEY (OID),
FOREIGN KEY (CUSTOMERS_ID) REFERENCES CUSTOMERS(ID)
);
```

Using Joins

- Now, let us insert data to the ORDERS table using the following query:

```
INSERT INTO ORDERS
VALUES (102,TO_DATE('08/October/2009','DD/MON/YY'), 3, 3000);
INSERT INTO ORDERS
VALUES (100,TO_DATE('08/October/2009','DD/MON/YY'), 3, 1500);
INSERT INTO ORDERS
VALUES (101,TO_DATE('20/November/2009','DD/MON/YY'), 2, 1560);
INSERT INTO ORDERS
VALUES (103,TO_DATE('20/May/2008','DD/MON/YY'), 4, 2060);
```

Using Joins

- Now, we have the following two tables – CUSTOMERS table and ORDERS table.

ID	NAME	AGE	ADDRESS	SALARY
1	Anil	32	Kathmandu	2000
2	Sandace	25	Pokhara	1500
3	Prashant	23	Chitwan	2000
4	Santosh	25	Biratnagar	6500
5	Shreeram	28	Lumbini	8500
6	Krishna	22	Lamjung	4500
7	Sonam	22	Bhojpur	10000

OID	DATE_OF_ORDER	CUSTOMERS_ID	AMOUNT
102	08-OCT-09	3	3000
100	08-OCT-09	3	1500
101	20-NOV-09	2	1560
103	20-MAY-08	4	2060

Using Joins

- Now, let us join these two tables in our SELECT statement using the following query:

```
SELECT ID, NAME, AGE, AMOUNT
FROM CUSTOMERS, ORDERS
WHERE  CUSTOMERS.ID = ORDERS.CUSTOMERS_ID;
```

- This would produce the following result:

ID	NAME	AGE	AMOUNT
2	Sandace	25	1560
3	Prashant	23	3000
3	Prashant	23	1500
4	Santosh	25	2060

Using Joins

- Here, it is noticeable that the join is performed in the WHERE clause.
- Several operators can be used to join tables, such as =, <, >, <>, <=, >=, !=, **BETWEEN**, **LIKE**, and **NOT**.
- They can all be used to join tables.
- However, the most common operator is the *equal to* symbol.

Different Types of Joins

JOIN	DESCRIPTION
INNER JOIN	Returns rows when there is a match in both tables.
LEFT JOIN	Returns all rows from the left table, even if there are no matches in the right table.
RIGHT JOIN	Returns all rows from the right table, even if there are no matches in the left table.
FULL JOIN	Returns rows when there is a match in one of the tables.
SELF JOIN	Is used to join a table to itself as if the table were two tables, temporarily renaming at least one table in the SQL statement.
CARTESIAN JOIN	Returns the Cartesian product of the sets of records from the two or more joined tables.

NULL Values

- The SQL **NULL** is the term used to represent a missing value.
- A NULL value in a table is a value in a field that appears to be blank.
- A field with a NULL value is a field with no value.
- It is very important to understand that a NULL value is different than a zero value or a field that contains spaces.
- The NULL value can cause problems when selecting data.
- However, because when comparing an unknown value to any other value, the result is always unknown and not included in the results.
- You must use the **IS NULL** or **IS NOT NULL** operators to check for a NULL value.

NULL Values – Example

- Consider the following CUSTOMERS table.

ID	NAME	AGE	ADDRESS	SALARY
1	Anil	32	Kathmandu	2000
2	Sandace	25	Pokhara	1500
3	Prashant	23	Chitwan	2000
4	Santosh	25	Biratnagar	6500
5	Shreeram	28	Lumbini	8500
6	Krishna	22	Lamjung	(null)
7	Sonam	22	Bhojpur	(null)

- Now, following is the usage of the IS NOT NULL operator:

SELECT ID, NAME, AGE, ADDRESS, SALARY

FROM CUSTOMERS

WHERE SALARY IS NOT NULL;

ID	NAME	AGE	ADDRESS	SALARY
1	Anil	32	Kathmandu	2000
2	Sandace	25	Pokhara	1500
3	Prashant	23	Chitwan	2000
4	Santosh	25	Biratnagar	6500
5	Shreeram	28	Lumbini	8500

NULL Values – Example

- Consider the following CUSTOMERS table.

ID	NAME	AGE	ADDRESS	SALARY
1	Anil	32	Kathmandu	2000
2	Sandace	25	Pokhara	1500
3	Prashant	23	Chitwan	2000
4	Santosh	25	Biratnagar	6500
5	Shreeram	28	Lumbini	8500
6	Krishna	22	Lamjung	(null)
7	Sonam	22	Bhojpur	(null)

- Now, following is the usage of the IS NULL operator:

```
SELECT  ID, NAME, AGE, ADDRESS, SALARY
FROM CUSTOMERS
WHERE SALARY IS NULL;
```

ID	NAME	AGE	ADDRESS	SALARY
6	Krishna	22	Lamjung	(null)
7	Sonam	22	Bhojpur	(null)

BETWEEN Operator

- The BETWEEN operator selects values within a given range.
- The values can be numbers, text, or dates.
- The BETWEEN operator is inclusive: begin and end values are included.
- Syntax:

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

BETWEEN Operator – Example

- Consider the CUSTOMERS table having the following records.
- The following SQL statement selects all NAME with a SALARY between 2000 and 6000.

```
SELECT * FROM CUSTOMERS
WHERE SALARY BETWEEN 2000
AND 6000;
```

- The following SQL statement selects all NAME with a SALARY not between 2000 and 6000.

```
SELECT * FROM CUSTOMERS
WHERE SALARY NOT BETWEEN
2000 AND 6000;
```

ID	NAME	AGE	ADDRESS	SALARY
1	Anil	32	Kathmandu	2000
2	Anil	25	Pokhara	1500
3	Prashant	23	Chitwan	2000
4	Prashant	25	Biratnagar	6500
5	Shreeram	28	Lumbini	8500
6	Krishna	22	Lamjung	4500
7	Sonam	22	Bhojpur	10000

ID	NAME	AGE	ADDRESS	SALARY
1	Anil	32	Kathmandu	2000
3	Prashant	23	Chitwan	2000
6	Krishna	22	Lamjung	4500

ID	NAME	AGE	ADDRESS	SALARY
2	Sandace	25	Pokhara	1500
4	Santosh	25	Biratnagar	6500
5	Shreeram	28	Lumbini	8500
7	Sonam	22	Bhojpur	10000

IN Operator

- The IN operator allows you to specify multiple values in a WHERE clause.
- The IN operator is a shorthand for multiple OR conditions.
- Syntax:

SELECT column_name(s)

FROM table_name

WHERE column_name IN (value1, value2, ...);

OR

SELECT column_name(s)

FROM table_name

WHERE column_name IN (SELECT Statement);

IN Operator – Example

- Consider the CUSTOMERS table having the following records.
- The following SQL statement selects all customers that are located in Kathmandu, Biratnagar, and Bhojpur:

```
SELECT * FROM Customers
WHERE ADDRESS IN ('Kathmandu', 'Biratnagar', 'Bhojpur');
```

- The following SQL statement selects all customers that are not located in Kathmandu, Biratnagar, and Bhojpur:

```
SELECT * FROM Customers
WHERE ADDRESS NOT IN ('Kathmandu', 'Biratnagar', 'Bhojpur');
```

ID	NAME	AGE	ADDRESS	SALARY
1	Anil	32	Kathmandu	2000
2	Anil	25	Pokhara	1500
3	Prashant	23	Chitwan	2000
4	Prashant	25	Biratnagar	6500
5	Shreeram	28	Lumbini	8500
6	Krishna	22	Lamjung	4500
7	Sonam	22	Bhojpur	10000

ID	NAME	AGE	ADDRESS	SALARY
1	Anil	32	Kathmandu	2000
4	Santosh	25	Biratnagar	6500
7	Sonam	22	Bhojpur	10000

ID	NAME	AGE	ADDRESS	SALARY
2	Sandace	25	Pokhara	1500
3	Prashant	23	Chitwan	2000
5	Shreeram	28	Lumbini	8500
6	Krishna	22	Lamjung	4500

END OF UNIT THREE