# RDBMS

## UNIT-1

### What is Relational 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.

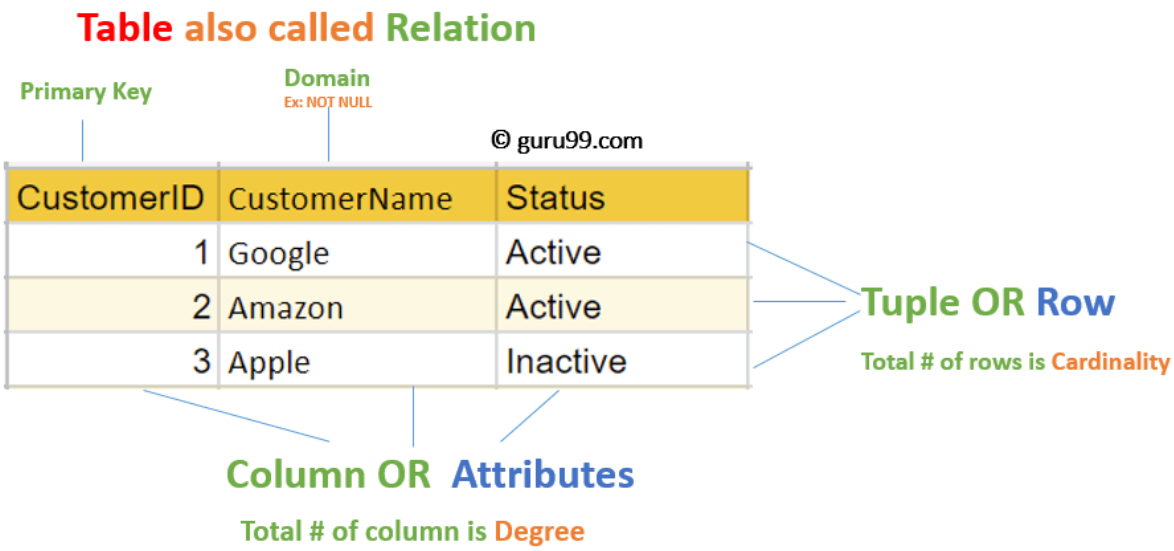
The table name and column names are helpful to interpret the meaning of values in each row. The data are represented as a set of relations. In the relational model, data are stored as tables. However, the physical storage of the data is independent of the way the data are logically organized.

Some popular Relational Database management systems are:

- DB2 and Informix Dynamic Server - IBM
- Oracle and RDB – Oracle
- SQL Server and Access - Microsoft

### Relational Model Concepts

1. **Attribute:** Each column in a Table. Attributes are the properties which define a relation. e.g., Student_Rollno, NAME,etc.
2. **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.
3. **Tuple** – It is nothing but a single row of a table, which contains a single record.
4. **Relation Schema:** A relation schema represents the name of the relation with its attributes.
5. **Degree:** The total number of attributes which in the relation is called the degree of the relation.
6. **Cardinality:** Total number of rows present in the Table.
7. **Column:** The column represents the set of values for a specific attribute.
8. **Relation instance** – Relation instance is a finite set of tuples in the RDBMS system. Relation instances never have duplicate tuples.
9. **Relation key** - Every row has one, two or multiple attributes, which is called relation key.
10. **Attribute domain** – Every attribute has some pre-defined value and scope which is known as attribute domain




**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 Relational database management system is mostly divided into three main categories are:

1. Domain constraints
2. Key constraints
3. Referential integrity constraints

**Domain Constraints**

Domain constraints can be violated if an attribute value is not appearing in the corresponding 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.

**Key constraints**

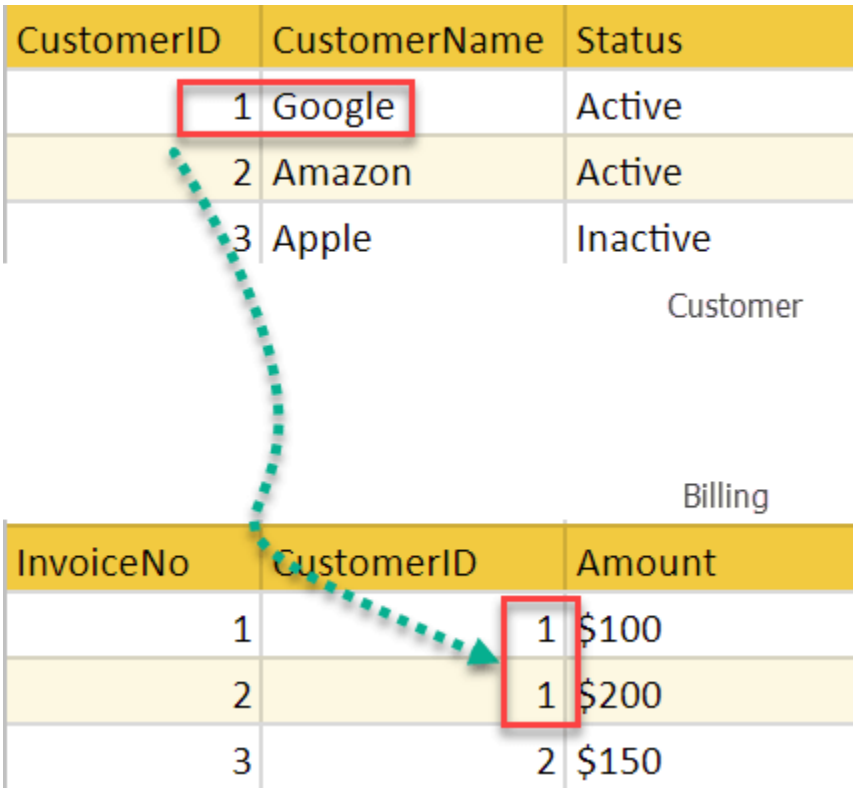
An attribute that can uniquely identify a tuple in a relation is called the key of the table. The value of the attribute for different tuples in the relation has to be unique.

**Referential integrity constraints**

Referential integrity constraints is base 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.

The diagram given below will show this concept.




In the above example, we have 2 relations, Customer and Billing.

Tuple for Customer ID =1 is referenced twice in the relation Billing. So we know Customer Name=Google has billing amount $300

**Operations in Relational Model**

Four basic update operations performed on relational database model are

Insert, update, delete and select.

- Insert is used to insert data into the relation
- Delete is used to delete tuples from the table.
- Modify allows you to change the values of some attributes in existing tuples.
- Select allows you to choose a specific range of data.

## Codd Rules :

Codd rules were proposed by E.F. Codd which should be satisfied by relational model.

1. **Foundation Rule:** For any system that is advertised as, or claimed to be, a relational data base management system, that system must be able to manage data bases entirely through its relational capabilities.
2. **Information Rule:** Data stored in Relational model must be a value of some cell of a table.
3. **Guaranteed Access Rule:** Every data element must be accessible by table name, its primary key and name of attribute whose value is to be determined.
4. **Systematic Treatment of NULL values:** NULL value in database must only correspond to missing, unknown or not applicable values.
5. **Active Online Catalog:** Structure of database must be stored in an online catalog which can be queried by authorized users.
6. **Comprehensive Data Sub-language Rule:** A database should be accessible by a language supported for definition, manipulation and transaction management operation.
7. **View Updating Rule:** Different views created for various purposes should be automatically updatable by the system.
8. **High level insert, update and delete rule:** Relational Model should support insert, delete, update etc. operations at each level of relations. Also, set operations like Union, Intersection and minus should be supported.
9. **Physical data independence:** Any modification in the physical location of a table should not enforce modification at application level.
10. **Logical data independence:** Any modification in logical or conceptual schema of a table should not enforce modification at application level. For example, merging of two tables into one should not affect application accessing it which is difficult to achieve.
11. **Integrity Independence:** Integrity constraints modified at database level should not enforce modification at application level.
12. **Distribution Independence:** Distribution of data over various locations should not be visible to end-users.
13. **Non-Subversion Rule:** Low level access to data should not be able to bypass integrity rule to change data.

## Relational Algebra

Relational algebra is a procedural query language, which takes instances of relations as input and yields instances of relations as output. It uses operators to perform queries. An operator can be either unary or binary. They accept relations as their input and yield relations as their output. Relational algebra is performed recursively on a relation and intermediate results are also considered relations.

The fundamental operations of relational algebra are as follows −

- Select
- Project
- Union

- Set different
- Cartesian product
- Rename

Select Operation (σ)

It selects tuples that satisfy the given predicate from a relation.

**Notation** − $\sigma_p(r)$

**For example** −

$\sigma_{\boldsymbol{subject} = "database"}(Books)$

**Output** − Selects tuples from books where subject is 'database'.

Project Operation (∏)

It projects column(s) that satisfy a given predicate.

Notation − $\prod_{A1, A_2, A_n} (r)$

Where $A_1$, $A_2$ , $A_n$ are attribute names of relation **r**.

Duplicate rows are automatically eliminated, as relation is a set

**For example** −

$\prod_{subject, author} (Books)$;

Union Operation (∪)

It performs binary union between two given relations .

**Notation** − r U s

Where **r** and **s** are either database relations or relation result set (temporary relation).

For a union operation to be valid, the following conditions must hold −

- **r**, and **s** must have the same number of attributes.
- Attribute domains must be compatible.
- Duplicate tuples are automatically eliminated.

$\prod_{author} (Books) \cup \prod_{author} (Articles)$

**Output** − Projects the names of the authors who have either written a book or an article or both.

Set Difference (−)

The result of set difference operation is tuples, which are present in one relation but are not in the second relation.

**Notation** − **r** − **s**

Finds all the tuples that are present in **r** but not in **s**.

$\prod_{author}$ (Books) − $\prod_{author}$ (Articles)

**Output** − Provides the name of authors who have written books but not articles.

Cartesian Product (X)

Combines information of two different relations into one.

**Notation** − r X s

Where **r** and **s** are relations and their output will be defined as −

r X s = { q t | q ∈ r and t ∈ s}

$\sigma_{author = 'saket'}$(Books X Articles)

**Output** − Yields a relation, which shows all the books and articles written by saket.

Rename Operation (ρ)

The results of relational algebra are also relations but without any name. The rename operation allows us to rename the output relation. 'rename' operation is denoted with small Greek letter **rho** *ρ*.

**Notation** − $\rho_x$ (E)

Where the result of expression **E** is saved with name of **x**.

Relational Calculus

In contrast to Relational Algebra, Relational Calculus is a non-procedural query language, that is, it tells what to do but never explains how to do it.

Relational calculus exists in two forms −

**Tuple Relational Calculus (TRC)**

Filtering variable ranges over tuples

**Notation** − {T | Condition}

Returns all tuples T that satisfies a condition.

**For example** −

{ T.name |  Author(T) AND T.article = 'database' }

**Output** − Returns tuples with 'name' from Author who has written article on 'database'.

**Domain Relational Calculus (DRC)**

In DRC, the filtering variable uses the domain of attributes instead of entire tuple values.

**Notation** −

$\{ a_1, a_2, a_3, ..., a_n \mid P (a_1, a_2, a_3, ... ,a_n)\}$

Where a1, a2 are attributes and P stands for formulae built by inner attributes.

**For example** −

{< article, page, subject > | ∈ Tutorials ∧ subject = 'database'}

**Output** − Yields Article, Page, and Subject from the relation Tutorials, where subject is database.

The expression power of Tuple Relation Calculus and Domain Relation Calculus is equivalent to Relational Algebra.

# UNIT-II

**Functional Dependency**

The functional dependency is a relationship that exists between two attributes. It typically exists between the primary key and non-key attribute within a table.

X → Y

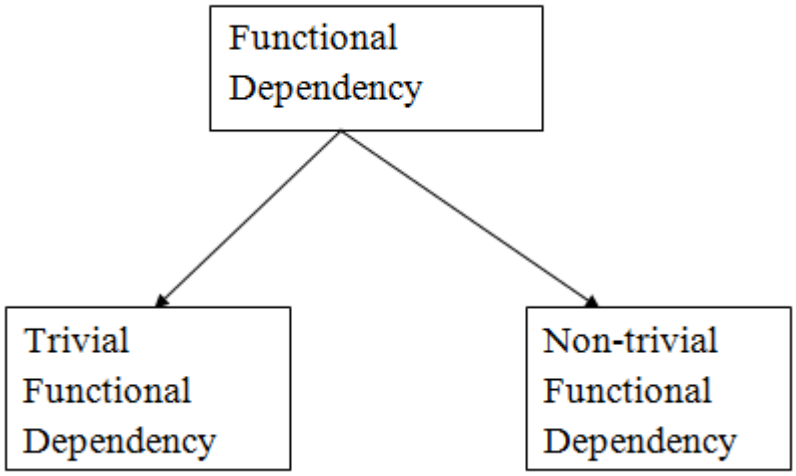
Assume we have an employee table with attributes: Emp_Id, Emp_Name, Emp_Address.

Here Emp_Id attribute can uniquely identify the Emp_Name attribute of employee table because if we know the Emp_Id, we can tell that employee name associated with it.

Functional dependency can be written as:

1. Emp_Id → Emp_Name

**Types of Functional Dependency**




1. Trivial functional dependency

- A → B has trivial functional dependency if B is a subset of A.
- The following dependencies are also trivial like: A → A, B → B

2. Non-trivial functional dependency

- A → B has a non-trivial functional dependency if B is not a subset of A.
- When A intersection B is NULL, then A → B is called as complete non-trivial.

**Characteristics of functional dependencies**:

1.In a functional dependency statement ,attributes present at left hand and right hand side should possess one-to-one relationship.

2.It hold for all the time.

3.Functional dependencies are non trivial.

**Transitive dependency**

A functional dependency is said to be transitive if it is indirectly formed by two functional dependencies. For e.g.

X -> Z is a transitive dependency if the following three functional dependencies hold true:

- X->Y
- Y does not ->X
- Y->Z

**Full functional Dependency :**

- A full functional dependency is a state of database normalization that equates to the normalization standard of Second Normal Form (2NF). In brief, this means that it meets the requirements of First Normal Form (1NF), and all non-key attributes are fully functionally dependent on the primary key.

**Normalization**: It 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.

Purpose of Normalization : 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.

**Anomalies in DBMS**

There are three types of anomalies that occur when the database is not normalized. These are – Insertion, update and deletion anomaly. Let’s take an example to understand this.

**Example**: Suppose a manufacturing company stores the employee details in a table named employee that has four attributes: emp_id for storing employee's id, emp_name for storing employee's name, emp_address for storing employee's address and emp_dept for storing the department details in which the employee works. At some point of time the table looks like this:

| emp_id | emp_name | emp_address | emp_dept |
|---|---|---|---|
| 101 | Rick | Delhi | D001 |
| 101 | Rick | Delhi | D002 |
| 123 | Maggie | Agra | D890 |
| 166 | Glenn | Chennai | D900 |
| 166 | Glenn | Chennai | D004 |

The above table is not normalized.

**Update anomaly**: In the above table we have two rows for employee Rick as he belongs to two departments of the company. If we want to update the address of Rick then we have to update the same in two rows or the data will become inconsistent. If somehow, the correct address gets updated in one department but not in other then as per the database, Rick would be having two different addresses, which is not correct and would lead to inconsistent data.

**Insert anomaly**: Suppose a new employee joins the company, who is under training and currently not assigned to any department then we would not be able to insert the data into the table if emp_dept field doesn't allow nulls.

**Delete anomaly**: Suppose, if at a point of time the company closes the department D890 then deleting the rows that are having emp_dept as D890 would also delete the information of employee Maggie since she is assigned only to this department.

To overcome these anomalies we need to normalize the data.

**Normalization**

Here are the most commonly used normal forms:

- First normal form(1NF)
- Second normal form(2NF)
- Third normal form(3NF)
- Boyce & Codd normal form (BCNF)

**First normal form (1NF)**

As per the rule of first normal form, an attribute (column) of a table cannot hold multiple values. It should hold only atomic values.

**Example**: Suppose a company wants to store the names and contact details of its employees. It creates a table that looks like this:

| emp_id | emp_name | emp_address | emp_mobile |
|---|---|---|---|
| 101 | Herschel | New Delhi | 8912312390 |
| 102 | Jon | Kanpur | 8812121212<br>9900012222 |
| 103 | Ron | Chennai | 7778881212 |

This table is **not in 1NF** as the rule says “each attribute of a table must have atomic (single) values”, the emp_mobile values for employees Jon & Lester violates that rule.

To make the table complies with 1NF we should have the data like this:

| emp_id | emp_name | emp_address | emp_mobile |
|---|---|---|---|
| 101 | Herschel | New Delhi | 8912312390 |
| 102 | Jon | Kanpur | 8812121212 |
| 102 | Jon | Kanpur | 9900012222 |
| 103 | Ron | Chennai | 7778881212 |
| 104 | Lester | Bangalore | 9990000123 |
| 104 | Lester | Bangalore | 8123450987 |

**Second normal form (2NF)**

A table is said to be in 2NF if both the following conditions hold:

- Table is in 1NF (First normal form)
- No non-prime attribute is dependent on the proper subset of any candidate key of table.

An attribute that is not part of any candidate key is known as non-prime attribute.

**Example**: Suppose a school wants to store the data of teachers and the subjects they teach. They create a table that looks like this: Since a teacher can teach more than one subjects, the table can have multiple rows for a same teacher.

| teacher_id | subject | teacher_age |
|---|---|---|
| 111 | Maths | 38 |
| 111 | Physics | 38 |
| 222 | Biology | 38 |
| 333 | Physics | 40 |
| 333 | Chemistry | 40 |

**Candidate Keys**: {teacher_id, subject}
**Non prime attribute**: teacher_age

The table is in 1 NF because each attribute has atomic values. However, it is not in 2NF because non prime attribute teacher_age is dependent on teacher_id alone which is a proper subset of candidate key. This violates the rule for 2NF as the rule says "**no** non-prime attribute is dependent on the proper subset of any candidate key of the table".

**Third Normal form (3NF)**

A table design is said to be in 3NF if both the following conditions hold:

- Table must be in 2NF
- Transitive functional dependency of non-prime attribute on any super key should be removed.

An attribute that is not part of any candidate key is known as non-prime attribute.

Or

The table is in 3 NF if either X is the candidate key or Y is a prime attribute.(if Functional dependency is of the form X→Y)

**Boyce Codd normal form (BCNF)**

It is an advance version of 3NF that's why it is also referred as 3.5NF. BCNF is stricter than 3NF. A table complies with BCNF if it is in 3NF and for every functional dependency X->Y, X should be the candidate key of the table.

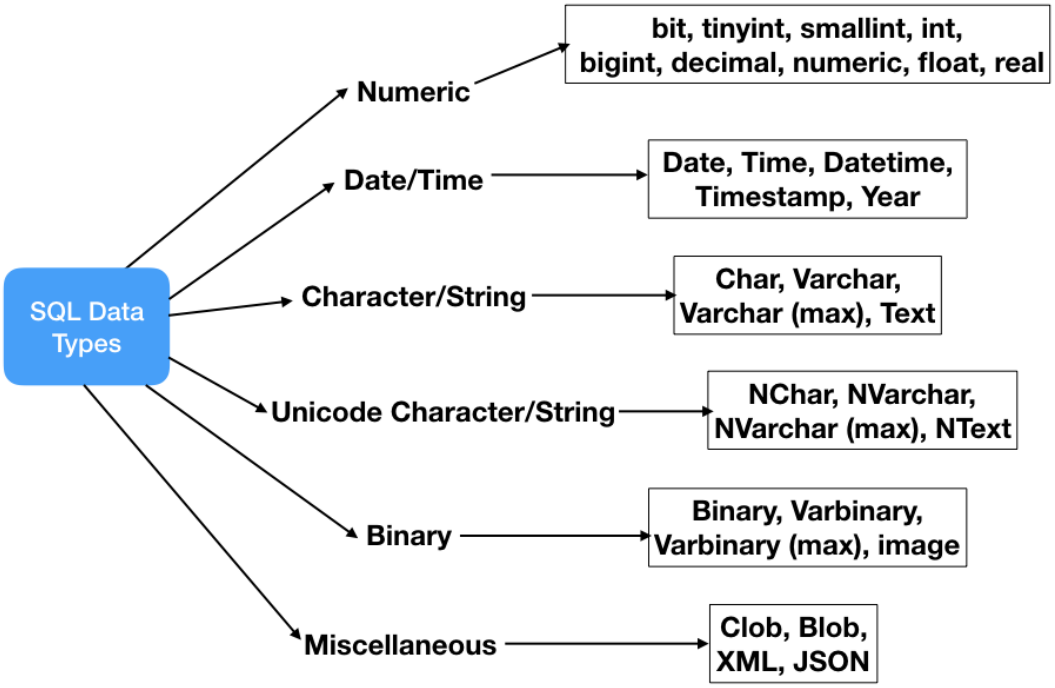
## UNIT-III

**What does SQL Mean :**

Structured Query Language (SQL) is a programming language that is typically used in relational database or data stream management systems.

It was developed by IBM in the early 1970s and is now an official standard recognized by the American National Standards Institute (ANSI) and the International Organization for Standardization (ISO).

**SQL Data Types :**




| | |
|---|---|
| **CHAR(Size)** | It is used to specify a fixed length string that can contain numbers, letters, and special characters.Its size can be 0 to 255 characters. Default is 1 |
| **VARCHAR(Size)** | It is used to specify a variable length string that can contain numbers, letters, and special Its size can be from 0 to 65535 characters. |
| **TEXT(Size)** | It holds a string that can contain a maximum length of 255 characters. |
| **BLOB(size)** | It is used for BLOBs (Binary Large Objects). It can hold up to 65,535 bytes. |

**SQL Constraints**

SQL constraints are used to specify rules for the data in a table.

Constraints 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 table. If there is any violation between the constraint and the data action, the action is aborted.

Constraints can be column level or table level. Column level constraints apply to a column, and table level constraints apply to the whole table.

The following constraints are commonly used in SQL:

- **NOT NULL** - Ensures that a column cannot have a NULL value
- **UNIQUE** - Ensures that all values in a column are different
- **PRIMARY KEY** - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table
- **FOREIGN KEY** - Uniquely identifies a row/record in another table
- **CHECK** - Ensures that all values in a column satisfies a specific condition
- **DEFAULT** - Sets a default value for a column when no value is specified
- **INDEX** - Used to create and retrieve data from the database very quickly

**Schema :**

A schema in a SQL database is a collection of logical structures of data. The schema is owned by a database user and has the same name as the database user. Security permissions can be applied to schemas hence schemas are an important tool for separating and protecting database objects on the basis of user access rights. It improves flexibility for security-related administration of the database.

**SQL Commands :**

- **Data Definition Language(DDL)** – Consists of commands which are used to define the database.
- **Data Manipulation Language(DML)** – Consists of commands which are used to manipulate the data present in the database.
- **Data Control Language(DCL)** – Consists of commands which deal with the user permissions and controls of the database system.
- **Transaction Control Language(TCL)** – Consist of commands which deal with the transaction of the database.

**Data Definition Language Commands (DDL) :**

Following are the basic DDL Commands.

- CREATE
- DROP
- TRUNCATE
- ALTER
- BACKUP DATABASE

**CREATE**

This statement is used to create a table or a database.

Syntax

```
CREATE DATABASE DatabaseName;
```

Example :

```
CREATE DATABASE Employee;
```

*The 'CREATE TABLE' Statement*

This statement is used to create a table.

Syntax

```
CREATE TABLE TableName (
Column1 datatype,
Column2 datatype,
Column3 datatype,
....

ColumnN datatype
);
```

Example :

```
CREATE TABLE Employee_Info (

EmployeeID int,

EmployeeName varchar(255),

Emergency ContactName varchar(255)

);
```

**DROP**

This statement is used to drop an existing table or a database.

Syntax

DROP DATABASE DatabaseName;

Example :

DROP DATABASE Employee;

**TRUNCATE**

This command is used to delete the information present in the table but does not delete the table. So once you use this command, your information will be lost, but not the table.

Syntax

TRUNCATE TABLE TableName;

Example :

TRUNCATE Table Employee_Info;

**ALTER**

This command is used to delete, modify or add constraints or columns in an existing table.

***The 'ALTER TABLE' Statement***

This statement is used to add, delete, modify columns in an existing table.

Syntax :

ALTER TABLE TableName
DROP COLUMN ColumnName;
Example :
ALTER TABLE Employee_Info
DROP COLUMN BloodGroup ;

**SQL INSERT INTO Syntax :**

The INSERT INTO statement is used to insert new records in a table.
INSERT INTO *table_name* (*column1*, *column2*, *column3*, ...)
VALUES (*value1*, *value2*, *value3*, ...);

Example :
INSERT INTO Customers (CustomerName, ContactName);
VALUES ('Saket', 'Tom', 'Delhi');

The SQL DELETE Statement

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

DELETE Syntax

DELETE FROM *table_name* WHERE *condition*;

Example :

DELETE FROM Customers WHERE CustomerName='neha';

The SQL UPDATE Statement

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

UPDATE Syntax

UPDATE *table_name*
SET *column1* = *value1*, *column2* = *value2*, ...
WHERE *condition*;

Example :

UPDATE Customers
SET ContactName = 'Shivam', City= 'Delhi'
WHERE CustomerID = 1;

**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.

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

Syntax :

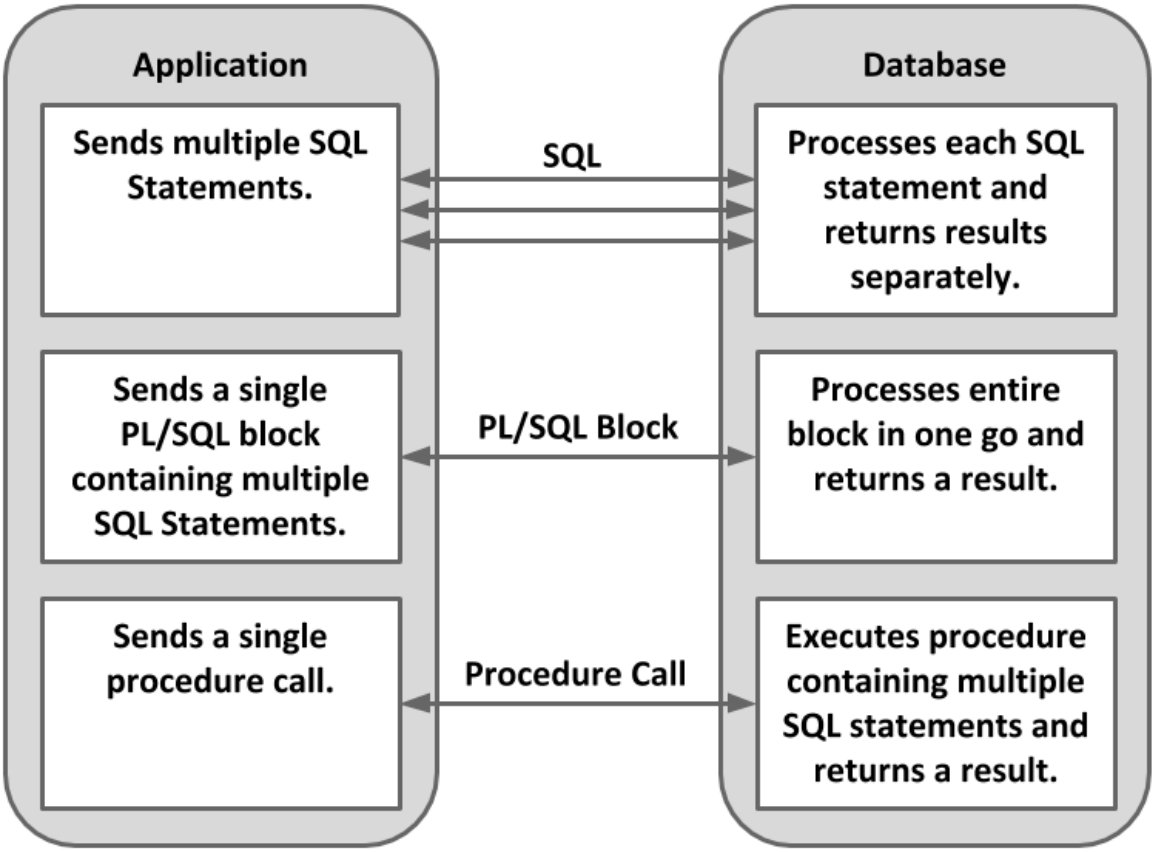
```
CREATE VIEW view_name AS
SELECT column1, column2.....
FROM table_name
WHERE condition;
```

**DELETING VIEWS :**
Drop VIEW view_name;

## UNIT-IV

**INTRODUCTION TO PL/SQL :**

- PL/SQL is a procedural extension of SQL, making it extremely simple to write procedural code that includes SQL as if it were a single language. In comparison, most other programming languages require mapping data types, preparing statements and processing result sets, all of which require knowledge of specific APIs.
- The data types in PL/SQL are a super-set of those in the database, so you rarely need to perform data type conversions when using PL/SQL. Ask your average Java or .NET programmer how they find handling date values coming from a database. They can only wish for the simplicity of PL/SQL.
- When coding business logic in middle tier applications, a single business transaction may be made up of multiple interactions between the application server and the database. This adds a significant overhead associated with network traffic. In comparison, building all the business logic as PL/SQL in the database means client code needs only a single database call per transaction, reducing the network overhead significantly.



**Advantage of Using PL/SQL**

1. Better performance, as SQL is executed in bulk rather than a single statement
2. High Productivity
3. Tight integration with SQL
4. Full Portability
5. Tight Security
6. Support Object Oriented Programming concepts.
7. Manageability.

**PL/SQL BLOCK :**

PL/SQL is a block-structured language whose code is organized into blocks. A PL/SQL block consists of three sections: declaration, executable, and exception-handling sections. In a block, the executable section is mandatory while the declaration and exception-handling sections are optional. A PL/SQL block has a name.
This is the basic structure of PL/SQL block.

8. 


**Declaration section**

A PL/SQL block has a declaration section where you declare variables, allocate memory for cursors, and define data types.

**2) Executable section** :

A PL/SQL block has an executable section. An executable section starts with the keyword BEGIN and ends with the keyword END. The executable section must have a least one executable statement, even if it is the NULL statement which does nothing.

**3) Exception-handling section** :

A PL/SQL block has an exception-handling section that starts with the keyword EXCEPTION. The exception-handling section is where you catch and handle exceptions raised by the code in the execution section.

**PL/SQL Execution Environment:**

The Oracle engine can process not only single SQL statement but also block of many statements. The call to Oracle engine needs to be made only once to execute any number of SQL statements if these SQL statements are bundled inside a PL/SQL block.

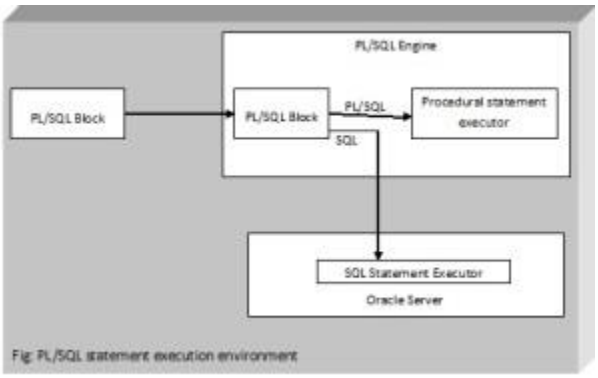


Fig: PL/SQL statement execution environment

The PL/SQL engine resides in the Oracle engine, the Oracle engine can process not only single SQL statements but also entire PL/SQL blocks.

These blocks are sent to the PL/SQL engine, where procedural statements are executed and SQL statements are sent to the SQL executor in the Oracle engine. Since the PL/SQL engine resides in the Oracle engine, this is an efficient and fast operation.

The call to the Oracle engine needs to be made only once to execute any number of SQL statements, if these SQL statements are bundled inside a PL/SQL block.

Since the oracle engine is called only once for each block, the speed of SQL statement execution is vastly enhanced, when compared to the Oracle engine being called once for each SQL sentence.

**Control Structures in PL/SQL :**

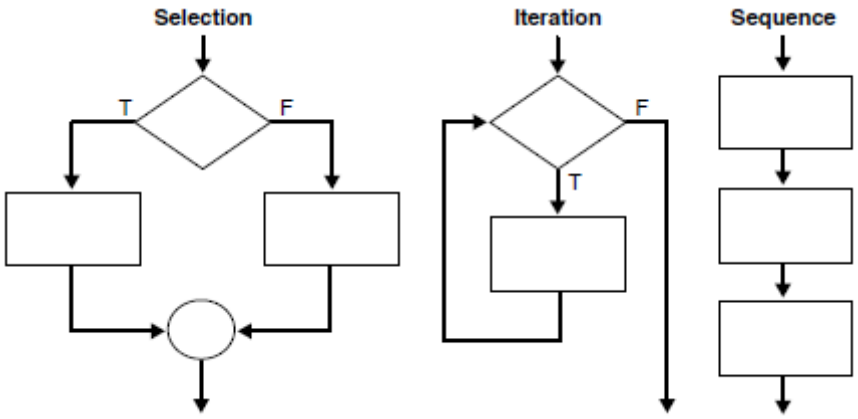
A condition is any variable or expression that returns a BOOLEAN value (TRUE or FALSE). The iteration structure executes a sequence of statements repeatedly as long as a condition holds true. The sequence-structure simply executes a sequence of statements in the order in which they occur.

Procedural computer programs use the basic control structures.

1.The selection structure tests a condition, then executes one sequence of statements instead of another, depending on whether the condition is true or false. A condition is any variable or expression that returns a BOOLEAN value (TRUE or FALSE).

2.The iteration structure executes a sequence of statements repeatedly as long as a condition holds true.

3.The sequence-structure simply executes a sequence of statements in the order in which they occur.




**PL/SQL CURSORS :** A cursor is a pointer to this context area. PL/SQL controls the context area through a cursor. A cursor holds the rows (one or more) returned by a SQL statement. The set of rows the cursor holds is referred to as the active set.

You can name a cursor so that it could be referred to in a program to fetch and process the rows returned by the SQL statement, one at a time. There are two types of cursors −

- Implicit cursors
- Explicit cursors

**Implicit Cursors :**

Implicit cursors are automatically created by Oracle whenever an SQL statement is executed, when there is no explicit cursor for the statement. Programmers cannot control the implicit cursors and the information in it.Whenever a DML statement (INSERT, UPDATE and DELETE) is issued, an implicit cursor is associated with this statement. For INSERT operations, the cursor holds the data that needs to be inserted. For UPDATE and DELETE operations, the cursor identifies the rows that would be affected.

**Explicit Cursors :**

Explicit cursors are programmer-defined cursors for gaining more control over the context area. An explicit cursor should be defined in the declaration section of the PL/SQL Block. It is created on a SELECT Statement which returns more than one row.

an explicit cursor includes the following steps −

- Declaring the cursor for initializing the memory
- Opening the cursor for allocating the memory
- Fetching the cursor for retrieving the data
- Closing the cursor to release the allocated memory

**PL/SQL TRIGGERS :** Triggers are stored programs, which are automatically executed or fired when some events occur. Triggers are, in fact, written to be executed in response to any of the following events −

- A **database manipulation (DML)** statement (DELETE, INSERT, or UPDATE)
- A **database definition (DDL)** statement (CREATE, ALTER, or DROP).

**PL/SQL CHARACTER SET :** A PL/SQL program consists of a sequence of statements, each made up of one or more lines of text. The precise characters available to you will depend on what database character set you're using. Every keyword in PL/SQL is made from various combinations of characters in the character set. By default, PL/SQL is a case-insensitive language. That is, uppercase letters are treated the same way as lowercase letters except when characters are surrounded by single quotes, which makes them a literal string.

Symbols in PL/SQL :

| Symbol | Description |
|---|---|
| ; | Semicolon: terminates declarations and statements |
| % | Percent sign: attribute indicator. Symbol used with the LIKE condition |
| _ | Single underscore: single-character wildcard symbol in LIKE condition |
| @ | At- sign |

PL/SQL Scalar Data Types :

| | |
|---|---|
| 1 | **Numeric**<br>Numeric values on which arithmetic operations are performed. |
| 2 | **Character**<br>Alphanumeric values that represent single characters or strings of characters. |
| 3 | **Boolean**<br>Logical values on which logical operations are performed. |
| 4 | **Datetime**<br>Dates and times. |

# RDBMS IMPORTANT QUESTIONS

Q1. Explain different functional dependency in detail with example?

Q2. Explain different Views of data in detail?

Q3. Write down the CODD's rules of relational model?

Q4. Difference between Client Server and Relational Model?

Q5. Explain the concept of relational algebra in detail with various algebraic operations?

Q6. Difference between Tuple relational and Domain relational calculus ?

Q7. Explain the concept of normalization with different Normal forms with example?

Q8. Difference between data redundancy and update anomaly?

Q9. Explain the basic concept of SQL?

Q10. What are the constraints in SQL?

Q11. Define Schema?

Q12.Write SQL query for selecting columns from a table named as student?

Q13. Write SQL query for outputting Sorted Data using Órder by Clause''?

Q14.Write SQL query to create a database named as ORG?

Q15.Write SQL command to show the database?

Q16.Write SQL command to create a table Worker having attributes Worker_id,Name,Salary?

Q17.Write SQL command to insert values into worker table?

Q18.Write SQL query to fetch unique values of DEPARTMENT from Worker table?

Q19. Write SQL query to delete record from customers table where id=6?

Q20. Explain the introduction of PL/SQl?

Q21. Write down the advantages of PL/SQL?

Q22.Explain the generic PL/SQL Block?

Q23. Explain PL/SQL execution environment?

Q24.Explain the control structures in PL/SQL?

Q25.Write SQL query to update table name as customer to change contact name to Riya and Country is India?

Q26.Write SQL query for outputting sorted data using ‘Örder by’ clause?

ANSWERS

Q20. PL/SQL is an extension of Structured Query Language (SQL) that is used in Oracle. Unlike SQL, PL/SQL allows the programmer to write code in a procedural format. Full form of PL/SQL is "Procedural Language extensions to SQL".

It combines the data manipulation power of SQL with the processing power of procedural language to create super powerful SQL queries.

PL/SQL means instructing the compiler 'what to do' through SQL and 'how to do' through its procedural way.

**Architecture of PL/SQL**

The PL/SQL architecture mainly consists of following three components:

1. PL/SQL block
2. PL/SQL Engine
3. Database Server

**Q21. Advantage of Using PL/SQL**

1. Better performance, as SQL is executed in bulk rather than a single statement
2. High Productivity
3. Tight integration with SQL
4. Full Portability
5. Tight Securitys
6. Support Object Oriented Programming concepts.
7. Manageability.
8. Scalability.

Q22. **PL/SQL** is a **block**-structured language whose code is organized into blocks. A PL/SQL block consists of three sections: declaration, executable, and exception-handling sections. In a block, the executable section is mandatory while the declaration and exception-handling sections are optional. A PL/SQL block has a name.

This is the basic structure of PL/SQL block.


**) Declaration section**

A PL/SQL block has a declaration section where you declare variables, allocate memory for cursors, and define data types.

**2) Executable section**

A PL/SQL block has an executable section. An executable section starts with the keyword BEGIN and ends with the keyword END. The executable section must have a least one executable statement, even if it is the NULL statement which does nothing.

**3) Exception-handling section**

A PL/SQL block has an exception-handling section that starts with the keyword EXCEPTION. The exception-handling section is where you catch and handle exceptions raised by the code in the execution section.

Q23. PL/SQL Execution Environment:

The Oracle engine can process not only single SQL statement but also block of many statements. The call to Oracle engine needs to be made only once to execute any number of SQL statements if these SQL statements are bundled inside a PL/SQL block.

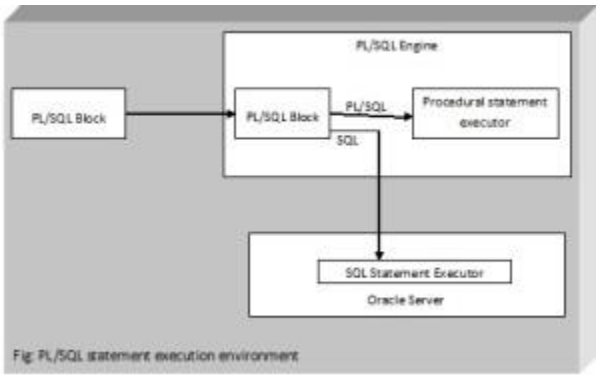


Fig: PL/SQL statement execution environment

The PL/SQL engine resides in the Oracle engine, the Oracle engine can process not only single SQL statements but also entire PL/SQL blocks.

These blocks are sent to the PL/SQL engine, where procedural statements are executed and SQL statements are sent to the SQL executor in the Oracle engine. Since the PL/SQL engine resides in the Oracle engine, this is an efficient and fast operation.

The call to the Oracle engine needs to be made only once to execute any number of SQL statements, if these SQL statements are bundled inside a PL/SQL block.

Since the oracle engine is called only once for each block, the speed of SQL statement execution is vastly enhanced, when compared to the Oracle engine being called once for each SQL sentence.

Q24.**Control Structures in PL/SQL :**

A condition is any variable or expression that returns a BOOLEAN value (TRUE or FALSE). The iteration structure executes a sequence of statements repeatedly as long as a condition holds true. The sequence-structure simply executes a sequence of statements in the order in which they occur.

Procedural computer programs use the basic control structures.

1.The selection structure tests a condition, then executes one sequence of statements instead of another, depending on whether the condition is true or false. A condition is any variable or expression that returns a BOOLEAN value (TRUE or FALSE).

2.The iteration structure executes a sequence of statements repeatedly as long as a condition holds true.

3.The sequence-structure simply executes a sequence of statements in the order in which they occur.