Database architecture

Data

- Data are known facts that can be recorded and that have implicit meaning.
- Eg . Reg number of student, name of the student, date of birth etc.

Database

- Is a collection of interrelated data which is used for the purpose of storage and retrieval of database operations .
- Eg. Telephone book, students record, railway reservation system

Data base management system

- Is a collection of programs that enables users to create and maintain a database.
- It is a general purpose software system that facilitates the processes of defining ,constructing ,manipulating and sharing database among various users and applications.

Charateristic of database approach

- To describe characteristic we need to compare traditional file processing and database approach

a) Self describing nature of database system

- Contains not only the database itself but also a complete definition or description of the database structure and constraints.
- Definition is stored in the DBMS catalog, which contain information such as structure of each file, type and storage format of each data item and various constraints on the data.
- Information stored in the catalog is called meta-data

b)Insulation between programs and data

- In traditional file processing , the structure of data files is embedded in the application programs so any changes to the structure of a file may require changing all programs that access that file.
- DBMS access programs do not require such changes in most cases . Structure of data-files is stored in the DBMS catalog separately from the access programs.

c)Support of multiple views of the data

- Database has many users,each of whom may require different perspective or view of the database

d)Sharing of data multiuser transaction processing

- Must allow multiple users to access the data base at the same time.

- Multiuser transaction is very hard to implement using the traditional file processing

Purpose of the database system

- Designed to deal with large volumes of data.
- Data management comprises both the construction of data storage systems and the provision of data manipulation methods.
- Must maintain the security of the information held despite system crashes or attempts by unauthorized access.

Goal of DBMS

- Transform data into information
- Transform information into knowledge
- Transform knowledge to the action

People associated with database system

- **Actors on the scene:**
  - **Database administrator**
  - **Database designers**
  - **End users**
  - **system analysts and application programmers(s/w developers)**

Database Administrators

- Responsible for authorizing access to the database ,co-ordinating and monitoring its use
- Acquiring software and hardware resources as needed .
- Accountable for problems such as security and poor system response time.

Database designers

- Identifying the data to be stored in the database
- Choosing appropriate structures to represent and store this data.
- Communicate with all prospective database users in order to understand their requirements and to create a design that meets these requirements .

End users

- Whose job require access to the database for querying ,updating and generating reports

Categories of end users

1.Casual end users: occasionally access the database ,who need different information each time.

2. Naïve or parametric end users:use a sizeable portion of database. Their job involves constantly querying and updating the database.using standard types of queries and updates called canned transaction(transactions that have been carefully programmed and tested.

- Eg. Bank tellers
- Reservation agents

3.Sophisticated end users

- Include engineers scientists, business analyst, and others who thoroughly familiarize themselves with the facilities of the DBMS

4.Standalone users

- Maintain personal databases by using ready-made program packages that provide easy to use menu based or graphics based interfaces.

5. System analyst and application programmers

- System analyst determine the requirements of end users, especially naïve and parametric end users and develop specification for standard canned transaction that meets these requirements.

- Application programmers implement these specification as programs ,then they test, debug and maintain these documents.

Workers behind the scene

- **DBMS system designers and implementers**: they design and implement the DBMS modules and interfaces as software package.
- **Tool developers:** design and implement tools. Tools are the software packages that facilitate database modelling and design ,database system design and improved performance.
- **Operators and maintenance personnel:**responsible for the actual running and maintenance of the hardware and software environment for the database

Data base schema

- **The logical and visual configuration of the entire relational database .**
- **Important to distinguish between the description of the data –base and database itself.**

- **Schema-describes the organization and storage of data in a database and defines the relationship between various tables**

Instances or database state

- Data in the database at a particular moment in time is called a database state or snapshot
- Also called the current set of occurrences or instances in the database.

Database architecture

- **DBMS describes the structure and how the users are connected to a specific database system.**
- **It affects the performance of the database as it helps to design, develop, implement, and maintain the database management system**.

The architecture of the database can be viewed as a logical model or physical model

- Logical model: the model describes the data and relationships within the database in terms of entities , attributes , and relationships.
- It provides a high-level view of the data and the relationships between data elements and is typically represented using and entity relationship diagram

- **physical model:**
- Describes the physical design and implementation of the database.

- It includes information about the storage structures, indexes and access paths used to store and retrieve data .
- Includes details about the hardware and software componentsused to support the database.

Types of database architecture

- 1 Tier Architecture
- 2 tier architecture
- 3-tier architecture

1 tier architecture

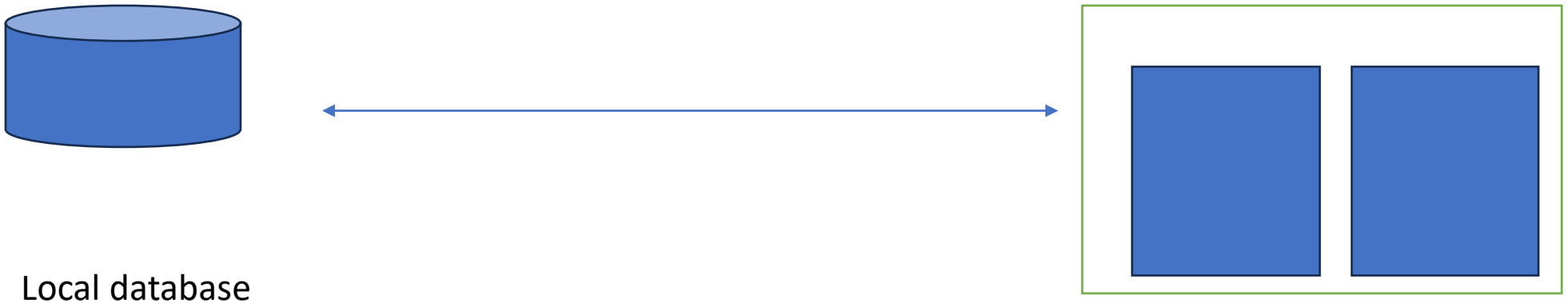
- All dbms components reside on a single server or platform, allowing direct access to the database by end users.
- It includes information about the storage structures, indexes and access paths used to store and retrieve data
- Direct connection results in rapid response times , making it a popular choice among programmers seeking to enhance local application.

- Edits made by the clients are reflected directly in the database in this structure, all processing is done on a single server.
- No network connection is required to execute database tasks.

1 tier architecture is suitable when

- The information is not updated frequently
- The database system is not being visited by many users
- We need a straightforward and simple means to edit or access the database for application development.

- Device

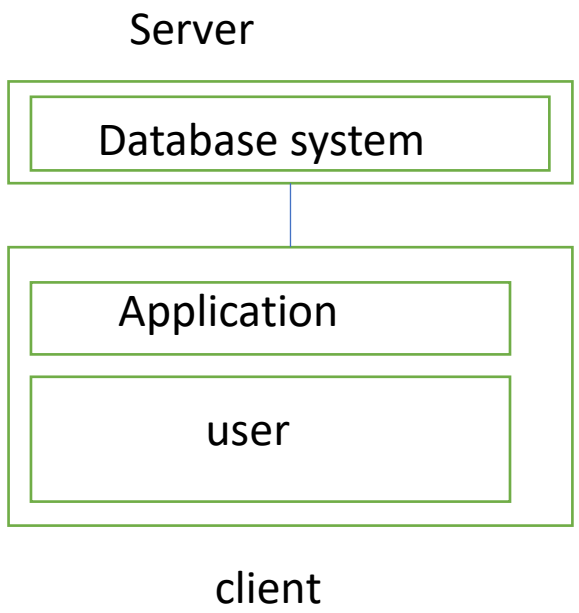


Local database

Application

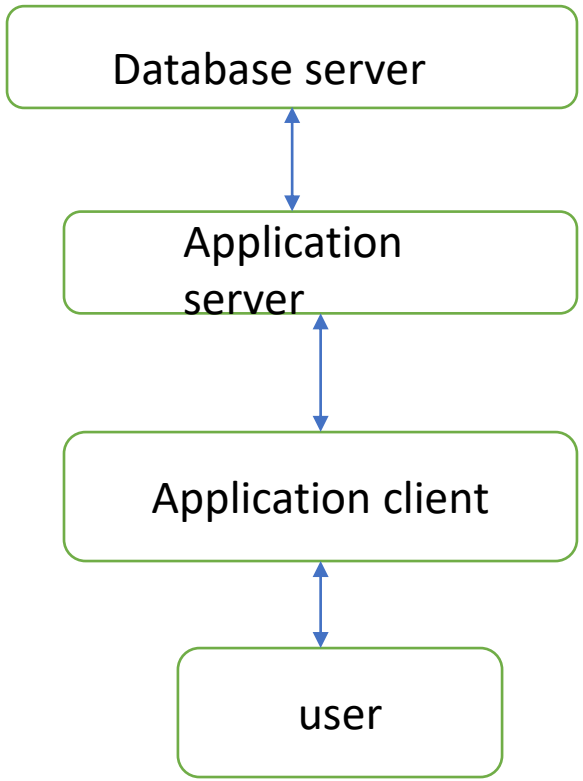
2- Tier Architecture

- Refers to a client-server architecture where the user interface and the application logic are separated into two separate components.
- The client component is typically the user interface and the server component is responsible for handling the data and business logic.
- The client component communicates directly with the server component to request data and perform actions.


Server
Database system
Application
user
client

3-tier architectre

- Another layer exists between the client and server in the 3-tier architecture.
- Client cannot communicate directly with the server with the design
- The program communicates with an application server, which then communicates with the database system.
- The end user has no knowledge of the database existence.
- Example: web application


Database server
Application server
Application client
user

Data independence

- THREE schema architecture can be used to further explain the concept of data independence
- Defined as the capacity to change the schema at one level of a database system without having to change the schema at the next higher level

Two types of data independence

- Logical data independence: the capacity to change the conceptual schema without having to change external schemas or application programs.

- Physical data independence: capacity to change the internal schema without having to change the conceptual schema,

Database languages

- A dbms has appropriate languages and interfaces to express database queries and updates.
- Database languages can be used to read store and update the data in database

Types of database languages

- **DDL(data definition language)**used to define database structure or pattern.
- Used to create schema, tables, indexes, constraints in database
- DDL is used to store the information of metadata like number of tables and schemas, their names, indexes, columns in each table, constraints.
- DDL commands: CREATE , ALTER, DROP, TRUNCATE, RENAME

- **DML (data manipulation language**): used for accessing and manipulating data in database
- Handles user request

- DML commands: SELECT, INSERT, DELETE, UPDATE.

- **DCL(data control language**) : used to retrieve the stored data.
- Mainly used used to revoke and grant the user access to database

- DCL commands: REVOKE , GRANT

- **TCL(Transaction control language):** used to run the changes made by the DML statement.
- Manages the transaction within the database.

- TCL commands: ROLLBACK,COMMIT

DATABASE INTERFACES

- Ability to input queries to a database without using the query language itself.

- User friendly interfaces provided by a DBMS includes:

- **Menu based interfaces for web clients or browsing**: these interfaces present the user with lists of options that lead the user through the formation of a request.
- Advantage of using is that they remove the tension of remembering specific commands and syntax of any query language.
- Pull down menus are a very popular technique in web based interfaces

- **Form based interfaces:** displays a form to each user.

  users can fill out all of the form entries to insert new data , or they can fill out only certain entries,

- **Graphic user interface**: displays a schema to the user in diagrammatic form.
- The user can then specify a query by manipulating the diagram.
- Natural language interface: accept request written in English or some other language and attempt to understand them.
- Refers to words in its schemas as well as to the set of standard words in a dictionary to interpret the request.

- **Speech input and output**: interfaces accept speech as an input and output
- Mostly used in the inquiry for telephone directory or to get flight info over the smart gadgets.
- **Interfaces for parametric users**: like bank tellers have a small set of operations that they must perform repeatedly.
- Perform a request with minimum keystrokes
- Used in bank transaction to deposit or withdraw of money

Hierarchical data model?

A hierarchical data model is a way of organizing and representing data in a hierarchical structure. In this model, data is organized into a tree-like structure with parent-child relationships. Each parent node can have multiple child nodes, but each child node has only one parent. This hierarchical arrangement is often used to represent relationships between different entities or elements in a system.

Example: A classic example of a hierarchical data model is an organizational chart, where the CEO is at the top (root), and various departments and employees form the branches of the tree.

Network Data Model?

The network data model is a database model that represents data as a collection of records, each connected to one or more neighboring records. Unlike the hierarchical model, which organizes data in a tree-like structure, the network model allows for more complex relationships, including many-to-many relationships. It was developed to address some of the limitations of the hierarchical model.

Example: An example of the network model is a university database where students can be associated with multiple courses, and courses can have multiple instructors. This allows for the representation of many-to-many relationships between students and courses, as well as between courses and instructors.

Relational Data Model?

The relational data model is a database model that organizes data into tables, where each table consists of rows and columns. It was introduced by E.F. Codd in 1970, and it has become the predominant model for managing and representing structured data in modern database management systems (DBMS).

Example: In a university database, you might have tables for students, courses, and instructors. The students' table could have columns like student_id, name, and major. Courses might have columns like course_id, course_name, and instructor_id, establishing a relationship between the courses and instructors.

Object-Oriented Data Model?

The Object-Oriented Data Model (OODM) is a database model that extends the principles of object-oriented programming (OOP) to database design. In this model, data is represented as objects, similar to how they are represented in programming languages like Java or C++. The Object-Oriented Data Model is designed to address the limitations of traditional relational databases in handling complex data structures and relationships.

Example: In a banking system, you might have classes for "Account" and "Customer," where each account object encapsulates data such as account number, balance, and associated customer. The classes might have methods like deposit and withdraw.

Object-Relational Data Model?

The Object-Relational Data Model (ORDBM) is an extension of the relational database model that incorporates object-oriented programming concepts. It aims to combine the strengths of both the relational and object-oriented paradigms, allowing for the representation of more complex data structures and relationships. The ORDBM attempts to bridge the gap between the relational data model and the object-oriented data model.

Example: In a library database, you might have an "Author" class with attributes like author_id and author_name. The "Book" class could inherit from the "Author" class and have additional attributes like book_id and title. Methods associated with these classes could include retrieving a list of books by a specific author.

Pl/sql

Pl/sql exception

- An error occurs during the program execution is called exception in pl/sql.pl/sql facilitates programmers to catch such exception using exception block in the program and an appropriate action is taken against the error condition.

Two types of exception

- System defined exception: which are executed when any database rule is violated by the programs

- User defined exception: a user defined exception can be raised explicitly ,using either a raise statement or the procedure DBMS_standard.Raise_Application_error.

- Declare
- <declaration section>
- Begin
- <executable command>
- Exception
- <exception handling goes here>
- When exception 1 then
- Exception1 handling statements

- When exception 2 then
- Exception 2 handling statements
- .
- .
- .when others then
- Exception handling statements
- .
- End;

ID	NAME	AGE	ADDRESS	SALARY
1	mohan	19	kerala	30000
2	yashaswini	19	karnataka	35000
3	jahanavi	19	andhra	40000
4	kaushik	19	hyderabad	45000
5	pavan	19	andhra	45000
6	tejashree	19	karnataka	30000

- Declare
- C_id customers.id%type:=8;
- C_name customers.name%type;
- C_addr customers.address%type;
- Begin
- Select name,address into c_name,c_addr
- From customers
- Where id=c_id;

- Dbms_output.put_line(‘name:’]] c_name);
- Dbms_output.put_line(‘address’]] c_addr);
- Exception
- When no_data_found then
- Dbms_output.put_line(‘no such customer’);
- When others then
- Dbms_output.put_line(‘error’);
- End;

output

- No such customer
- Why???
- Since there is no customer id value 8 in our database

Triggers

- Triggers are stored program, which are automatically executed or fired when some event occurs, triggers are written to executed in response to any of the following events.

- A DML statement(DELETE,INSERT,UPDATE)
- A DDL statement(CREATE,ALTER,or DROP)
- A database operation (SERVERERROR,LOGON,LOGOFF,STARTUP,or SHUTDOWN)

- Triggers could be defined on the table ,view,schema or database with which the event is associated

- Create [or replace]trigger trigger_name
- {Before|after|instead of}
- {insert[or]|update[or]|delete}
- [of col_name]
- On table_name
- [referencing old as o new as n]
- [for each row]
- When (condition)

- DECLARE
- declaration statement
- BEGIN
- Executable statements
- EXCEPTION
- exception handling statements
- END;

- Create [or replace]trigger trigger_name: it creates or replaces an existing trigger with the trigger_name.

- {BEFOREAFTER|INSTEAD OF }: this specifies when the trigger would be executed.
- The **INSTEAD OF** clause is used for creating trigger on a view.
- {INSTEAD[OR]| UPDATE[OR]|DELETE}: this specifies DML operation.
- [of col_name]: specifies the column name that would be updated

- [ON table_name]: specifies the name of the table associated with the trigger.
- [referencing old as o new as n]: this allows you to refer new and old values for various DML statements,like INSERT,UPDATE and DELETE.
- [for each row]: this specifies a row level trigger , the trigger would be executed for each row being affected. Otherwise the trigger will execute just once when the sql statement is executed, which is calleda table level trigger.

- When (condition): this condition provides a condition for rows for which the trigger would fire. This clause is valid only for row level triggers

ID	NAME	AGE	ADDRESS	SALARY
1	mohan	19	kerala	30000
2	yashaswini	19	karnataka	35000
3	jahanavi	19	andhra	40000

Create row level trigger for the customers table that would fire for INSERT or UPDATE or DELETE on the customer table.

- Create [or replace]trigger display_salary_changes
- Before DELETE or INSERT or UPDATE ON customer
- for each row
- When (NEW.id>0)

- DECLARE
- sal_diff numer;
- BEGIN
- Sal_diff:=:new.salary:old.salary;
- Dbms_output.put_line(old salary://: old.salary);
- Dbms_output.put_line(new salary://: new.salary);
- Dbms_output.put_line(salary difference://: sal_diff);
- End;

output

- Trigger created

- Use the following code to get the old salary and salary difference after the trigger created.

```
DECLARE
   total_rows number(2);
BEGIN
Update customer
Set salary=salary+3000;
If sql%notfound then
Dbms_output.put_line(no customer updated);
Elsif sql %found then
Total_rows:=sql%rowcount;
Dbms_output.put_line(total_rows//customers updted);

End if;
END;
```

output

- Old salary:30000
- New salary:33000
- Salary difference: 3000
- Old salary:xxxxx
- New salary:xxxxx
- Salary difference: xxxxx
- Old salary:xxxxx
- New salary:xxxxx
- Salary difference: xxxxx
- 3 customers updated

Important questions

- What is sql
- What is relational algebra
- What is DML ? Give example
- What is relational schema and relational instance? Give one example
- What is view and its advantages
- What is trigger
- Explain selection and projection operation in relational algebra with an example

- Explain the following SQL commands with example

i) create ii)Alter iii) Select iv) Truncate.

- Expalin pl/sql control statements with an example
- Write pl/sql code to find the factorial of a given number
- Explain the concept of commit and ROLLBACK
- Define referential integrity constraints with example
- List out the different types of join operation.
- What is group by clause? Give example.

- Mention the kind of constraints we can specify in the create command DDL
- What are the advantages of PL/SQL
- What is a constraint? Give the detailed explanation of key constraint and domain constraint
- Explain selection and projection operation in relational algebra with an example
- What is cursor? What are the cursor attributes ?explain

- Explain for ..loop statements in pl/sql with an example.
- What are the application of relational algebra in RDBMS
- What is an exception?mention major types of exceptions
- Explain briefly DL statements with syntax and examples.
- What is database trigger?explain four types of trigger.
- Explain while …loop statements in pl/sql with an example.
- Expalin Cartesian product and selection operation
- expalin relational algebra in detail

Data normalization

introduction

- A database with poorly designed tables and relationships will fail to meet the needs of the users.
- Normalization is a process of organising the data and attributes of a database.
- Done to reduce data redundancy in a database and to ensure logical data storage

Data redundancy

- Refers to having the same data in multiple locations.

Relational data model

- Is a mathematical concept based on the ideas of sets.
- A relation is the core of the relational model of data Dr. E.F.Codd of IBM first proposed the concept in the paper in 1970. “ A Relational model for large shared data banks” communications of the ACM ,june 1970

Relation name: student table

attributes

tuples

Regno	Name	Gender	Dob	course
101	raneet	m	20-sep-2005	bca
102	kaushik	m	20-sep-2005	bcom
103	monika	f	20-sep-2005	bba
104	srilakshmi	f	20-sep-2005	bca

- Relation: it is a table which has rows and columns in the data model, where rows represent cords and columns represent the attributes

- Tuples: a single row in a database that contains a single record for that relation.
- Attributes: table column are referred to as attributes of the relation.
- Cardinality of a relation: the number of tuples in a relation determines its cardinality. In this case the relation has cardinality of 4

- Degree of a relation: each column in the tuple is called an attribute. The number of attributes in a relation determines its degree. The relation has a degree of 5.
- Domain: the type of data that the attribute represents is defined by the domain.

- A domain is particularly the set of all possibilities that an attribute may validly contain. It is improper to corelate domains with data type.
- Domain is a logical concept whereas datatype is a physical one.
- ‘Age’ is a domain and ‘number’ is a datatype

- Properties of a relation:

A relation with N columns and M rows is said to be of degree N and Cardinality M.

Below Student table which shows the relation of degree 3 and cardinality 5

Regno	Name	course
101		
102		
103		
104		
105		

- All entries in a given column are of the same kind or type.
- Attributes are unordered– the order of columns in a relation is meaningless. When displaying a relation in tabular form,columns can be arranged in any order

Unit - 3

RELATIONAL MODEL CONCEPTS 1

1.1 INTRODUCTION

Relational Model was proposed by E.F. Codd to model data in the form of relations or tables. After designing the conceptual model of Database using ER diagram, we need to convert the conceptual model in the relational model which can be implemented using any RDBMS languages like Oracle SQL, MySQL etc.

What is Relational Model?

Relational Model represents how data is stored in Relational Databases. A relational database stores data in the form of relations (tables). Consider a relation STUDENT with attributes ROLL_NO, NAME, ADDRESS, PHONE and AGE shown in Table:

ROLL_NO	NAME	ADDRESS	PHONE	AGE
1	SURYA	KOLAR	9454678998	19
2	RAM	BANGALORE	9890234549	18
3	REKHA	MANGALORE		19
4	ANAND	UDUPI	9667584833	20

IMPORTANT TERMINOLOGIES

Attribute: Attributes are the properties that define a relation. e.g.; **ROLL_NO, NAME**

Relation Schema: A relation schema represents name of the relation with its attributes. e.g.; STUDENT (ROLL_NO, NAME, ADDRESS, PHONE and AGE) is relation schema for STUDENT. If a schema has more than 1 relation, it is called Relational Schema.

Tuple: Each row in the relation is known as tuple. The above relation contains 4 tuples, one of which is shown as:

1	SURYA	KOLAR	9454678998	19

Domain:: A domain is referred to in a relation schema by the **domain name** and has a set of associated values.

Example: Students (Roll_No: integer, *Name*: string, *Address*: string, Phone: integer, *Age*: integer).

This says that the field named **'Name'** has a domain named 'string'.

The set of values associated with domain 'string' is the set of all character strings.

Relation Instance: The set of tuples of a relation at a particular instance of time is called as relation instance. Table 1 shows the relation instance of STUDENT at a particular time. It can change whenever there is insertion, deletion or updation in the database.

Degree: The number of attributes in the relation is known as degree of the relation. The **STUDENT** relation defined above has degree 5.

Cardinality: The number of tuples in a relation is known as cardinality. The **STUDENT** relation defined above has cardinality 4.

Column: Column represents the set of values for a particular attribute. The column **ROLL_NO** is extracted from relation STUDENT.

ROLL_NO
1
2
3
4

NULL Values: The value which is not known or unavailable is called NULL value. It is represented by blank space. e.g.; PHONE of STUDENT having ROLL_NO **2** is NULL.

1.2 CHARACTERISTICS OF RELATIONS

Ordering of Tuples in a Relation: A relation is defined as a set of tuples. The definition of a relation does not specify any order. For example, tuples in the STUDENT relation could be ordered by values of NAME, ROLL_NO, AGE, or some other attribute. There is no preference for one ordering over another. However, in a file, records are physically stored on disk (or in memory), so there always is an order among the records. This ordering indicates first, second, i^{th}, and last records in the file.

Ordering of attributes in a relation R: The ordering of attributes is not important, because the attribute name appears with its value. Attributes in R(A1, A2, ..., An) and the values in t=<v1, v2, ..., vn> to be *ordered*.

Values in a tuple: All values are considered *atomic* (indivisible). A special **null** value is used to represent values that are unknown or inapplicable to certain tuples.

1.3 CATEGORIES OF CONSTRAINTS

Constraints are a set of rules. It is used to maintain the quality of information. Constraints in the database can be categorized into 3 main categories:

1. Implicit constraints
2. schema-based constraints
3. semantic constraints or Application Based Constraints

describes the dq & storage of data & defines the relationship b/w various tables

1. **Implicit constraints:** Constraints that are applied in the data model is called **Implicit constraints.** Example: The constraint that a relation cannot have duplicate tuples is an implicit constraint.
2. **Schema-based constraints:** Constraints that are directly applied in the schemas of the data model, by specifying them in the DDL. These are called as **schema-based constraints or Explicit constraints**. The schema-based constraints are as follows:
   - **Domain** constraint
   - **Key** constraints
   - **Entity integrity** constraints
   - **Referential integrity** constraints
   - Constraints on **Nulls**
3. **Semantic constraints or Application Based Constraints:** Constraints that cannot be directly applied in the schemas of the data model. We call these Application based or **semantic constraints.**

   Another important category of constraints is **data dependencies**, which include functional dependencies and multivalued dependencies. They are used mainly in **normalization.**

1.4 RELATIONAL MODEL CONSTRAINTS

Mainly Constraints on the relational database are of 4 types:

- Domain constraints
- Key constraints
- Entity Integrity constraints
- Referential integrity constraints.

1.4.1 Domain constraint

- Domain constraints specify the **set of possible values** that may be associated with an attribute.
- Such constraints may also prohibit the use of **null** values for particular attributes.
- The data types associated with domains typically include standard numeric data types for integers, string, real, time, date, currency, etc

ID	NAME	SEMESTER	AGE
1000	Tom	1st	17
1001	Johnson	2nd	24
1002	Leonardo	5th	21
1003	Kate	3rd	19
1004	Morgan	8th	A

Not allowed, because **AGE** is an integer attribute

In above table, the fifth record **AGE** attribute has value **'A'**. This violates domain constraints, because **AGE** is an integer attribute.

1.4.2 Key Constraints or Uniqueness Constraints

- These are called uniqueness constraints since it ensures that every tuple in the relation should be **unique**.
- A relation can have **multiple keys** or **candidate keys**(minimal superkey), out of which we choose one of the keys as primary key.
- We don't have any restriction on choosing the primary key out of candidate keys, but it is suggested to go with the candidate key with less number of attributes.
- **Null** values are not allowed in the primary key, hence Not Null constraint is also a part of key constraint.

ID	NAME	SEMESTER	AGE
1000	Tom	1st	17
1001	Johnson	2nd	24
1002	Leonardo	5th	21
1003	Kate	3rd	19
1002	Morgan	8th	22

Not allowed, because all row must be unique

In above table, the attribute ID is the primary key. The ID column contain 1002 twice, hence it violates **key constraints**

1.4.3 Entity Integrity Constraints

- The entity integrity constraint states that primary key value can't be null.
- This is because the primary key value is used to identify individual rows in relation and if the primary key has a null value, then we can't identify those rows.
- A table can contain a null value other than the primary key field

ID	NAME	SEMESTER	AGE
1000	Tom	1st	17
1001	Johnson	2nd	24
1002	Leonardo	5th	21
1003	Kate	3rd	19
	Morgan	8th	22

Not allowed as primary key can't contain a NULL

In above table, the attribute ID is the primary key. In fifth record, the ID column contain NULL value. hence it violates **Entity Integrity constraints**

1.4.4 Referential Integrity Constraints

- The Referential integrity constraints is specified between two relations or tables and used to maintain the consistency among the tuples in two relations.
- This constraint is enforced through foreign key, when an attribute in the foreign key of relation R1 have the same domain(s) as the primary key of relation R2, then the foreign key of R1 is said to reference or refer to the primary key of relation R2.
- The values of the foreign key in a tuple of relation R1 can either take the values of the primary key for some tuple in relation R2, or can take NULL values, but can't be empty.

Table 1

EMP_ID	NAME	AGE	D_No
1	Jack	20	11
2	Harry	40	24
3	John	27	**18**
4	David	38	13

← **Foreign Key**

← Not Allowed as D_No is not defined as a primary key of table 2 and in table 1, D_No is a foreign key

Table 2

Primary key →

D_No	D_Location
11	Mumbai
24	Delhi
13	Noida

The **Table 1** contain attributes such as **EMP_ID, NAME, AGE, D_NO,** where **D_NO** is a foreign key. Another table with two attributes such as **D_NO, D_Location** where **D_NO** is the primary key. In Table1, the third tuple **D_No** '18' is not allowed as that is not inserted in Table2.

Constraints on NULL values: Another constraint on attributes specifies whether null values are or are not permitted. For example, if every STUDENT tuple must have a valid, nonnull value for the Name attribute, then Name of **STUDENT** is constrained to be **NOT NULL**

Other Types of Constraints: Semantic Integrity Constraints can be specified and enforced within the application programs that update the database

Example:

- The salary of an employee should not exceed the salary of the employee's supervisor
- The max. no. of hours per employee for all projects he or she works on is 56 hrs per week

SQL-99 allows triggers and **ASSERTIONS** to express *semantic integrity constraints*

REVIEW QUESTIONS

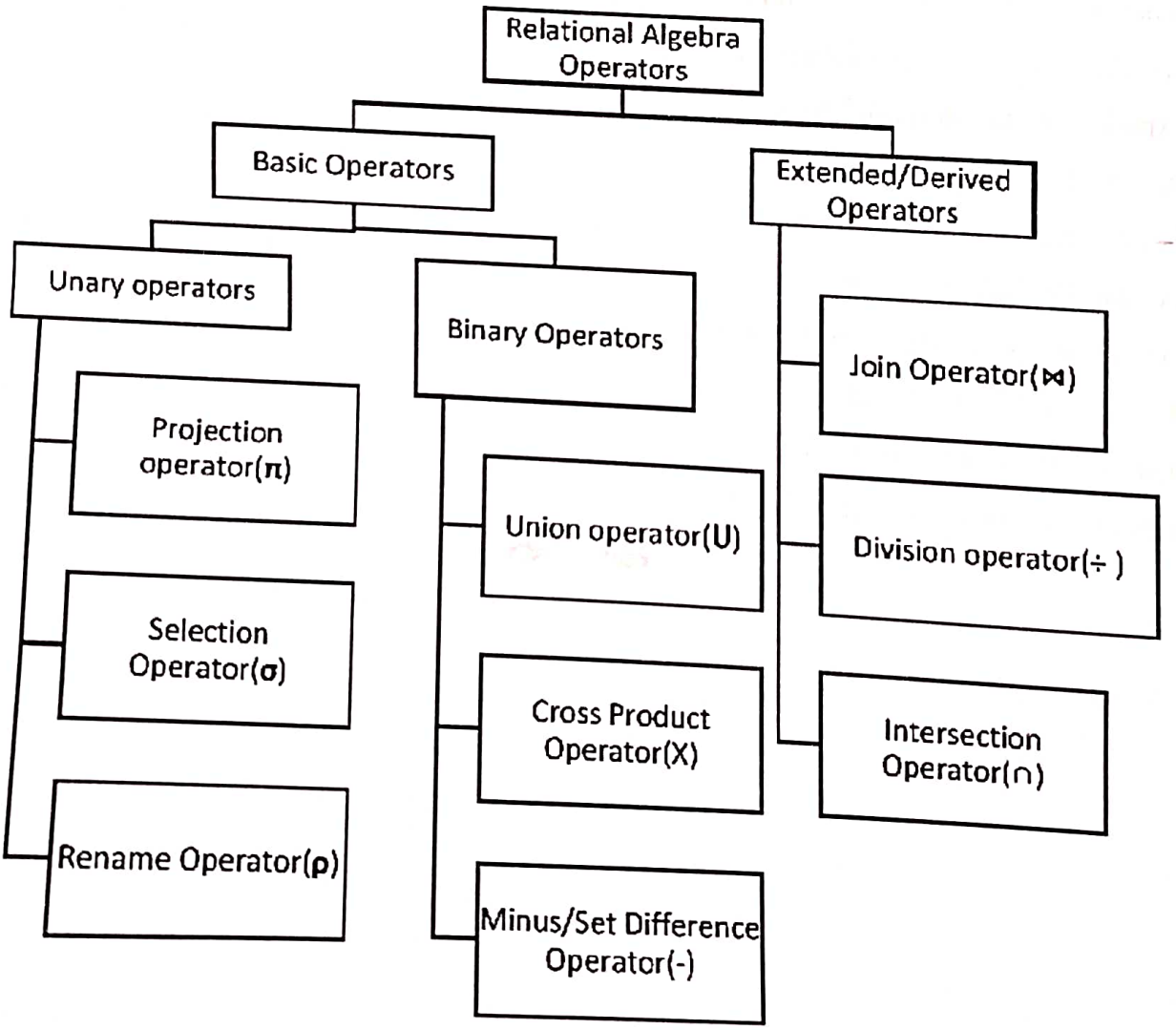
1. What is Domain, Attribute, Tuple?
2. List the characteristics of relations
3. Explain Relational model constraints?
4. Discuss Key constraints?
5. Explain Referential constraints?
6. Explain Domain constraints?
7. Explain schema based constraints?
8. What is NULL constraints?
9. Discuss integrity constraints?
10. Explain categories of constraints?

RELATIONAL ALGEBRA 2

2.1 INTRODUCTION

Relational algebra is a procedural query language, which takes relation as input and generate relation as output. Relational algebra mainly provides theoretical foundation for relational databases and SQL.

The basic set of operations for the relational model is the relational algebra. Relational Operators always work on one or more relational tables. Relational Operators always produce another relational table. The table produced by a relational operator has all the properties of a relational model. The operators in relational algebra are classified as



2.2 UNARY OPERATORS

Selection Operator: Selection Operator (σ) is a unary operator in relational algebra that performs a selection operation. It selects those rows or tuples from the relation that satisfies the selection condition.

Syntax

σ<selection_condition>(R)

Examples: Consider the following Student relation-

ID	Name	Subject	Age
100	Ashish	Maths	19
200	Rahul	Science	20
300	Naina	Physics	20
400	Sameer	Chemistry	21

Select tuples from a relation "Student" where subject is "database"

σsubject = "Science" (Student)

ID	Name	Subject	Age
200	Rahul	Science	20

Select tuples from a relation "Books" where subject is "Physics" and Age is "20"

σsubject = "Physics" Age = "20" (Student)

ID	Name	Subject	Age
300	Naina	Physics	20

Projection Operator: Projection Operator (π) is a unary operator in relational algebra that performs a projection operation. It displays the columns of a relation or table based on the specified attributes.

Syntax

π<attribute list>(R)

Example

Consider the following Student relation-

ID	Name	Subject	Age
100	Ashish	Maths	19
200	Rahul	Science	20
300	Naina	Physics	20
400	Sameer	Chemistry	21

Result for Query **πName, Age(Student)**

Name	Age
Ashish	19
Rahul	20
Naina	20
Sameer	21

Result for Query **πID , Name(Student)**

ID	Name
100	Ashish
200	Rahul
300	Naina
400	Sameer

Selection Operator performs horizontal partitioning of the relation. **Projection operator** performs vertical partitioning of the relation.

- There is only one difference between projection operator of relational algebra and SELECT operation of SQL.
- Projection operator does not allow duplicates while SELECT operation allowsduplicates.
- To avoid duplicates in SQL, we use "distinct" keyword and write SELECT distinct.
- Thus, projection operator of relational algebra is equivalent to SELECT operationof SQL.

Rename(ρ): Rename operator is used to giving another name to a relation.

Syntax:

ρ(Relation2, Relation1)

To rename STUDENT relation to STUDENT1, we can use rename operator like:

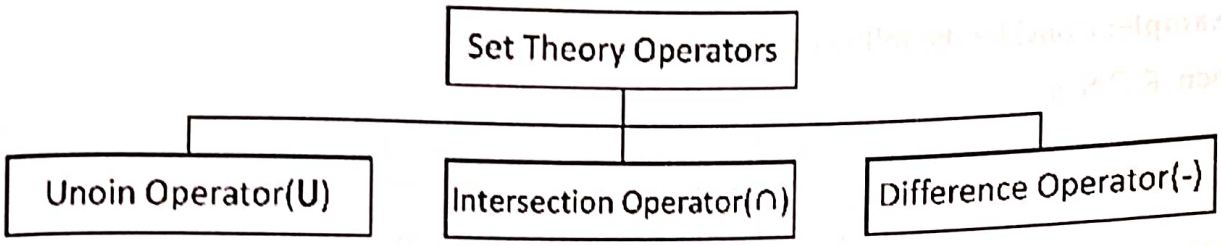
ρ(STUDENT1, STUDENT)

If you want to create a relation STUDENT_NAMES with ROLL_NO and NAME from STUDENT, it can be done using rename operator as:

ρ(STUDENT_NAMES, $\prod_{(ROLL\_NO, NAME)(STUDENT)}$**)**

2.3 SET THEORY OPERATORS

Following operators are called as set theory operators



Condition For Using Set TheoryOperators

To use set theory operators on two relations, the two relations must be union compatible.

Union compatible property means

- Both the relations must have same number of attributes.
- The attribute domains (types of values accepted by attributes) of both the relations must be compatible.

Union Operator (U): Let R and S be two relations.Then R ∪ S is the set of all tuples belonging to either R or S or both. In R ∪ S, duplicates are automatically removed. Union operation is both commutative and associative.

Example: Consider the following two relations R and S

Relation R

ID	Name	Subject
100	Ankit	English
200	Pooja	Maths
300	Komal	Science

Relation S

ID	Name	Subject
100	Ankit	English
400	Pooja KAJOL	French

Then, R ∪ S is

ID	Name	Subject
100	Ankit	English
200	Pooja	Maths
300	Komal	Science
400	Kajol	French

Intersection Operator (∩)

Let R and S be two relations. Then R ∩ S is the set of all tuples belonging to both R and S. In R ∩ S, duplicates are automatically removed. Intersection operation is both commutative and associative.

Example: Consider the following two relations R and S

Then, R ∩ S is

ID	Name	Subject
100	Ankit	English

Difference Operator (-): Let R and S be two relations. Then R – S is the set of all tuples belonging to R and not to S. In R – S, duplicates are automatically removed. Difference operation is associative but not commutative.

Example: Consider the above two relations R and S, Then, R – S is-

ID	Name	Subject
200	Pooja	Maths
300	Komal	Science

2.4 CARTESIAN PRODUCT (X)

Cross product between two relations let say A and B, so cross product between A X B will results all the attributes of A followed by each attribute of B. Each record of A will pairs with every record of B.

Table A

Name	Subject	Sex
Ram	24	Male
Abhi	22	Female
Keerthana	23	Female

Table B

ID	Course
1	JAVA
2	DS

Name	Age	Sex	ID	Course
Ram	24	Male	1	JAVA
Ram	24	Male	2	DS
Abhi	22	Female	1	JAVA
Abhi	22	Female	2	DS
Keerthana	23	Female	1	JAVA
Keerthana	23	Female	2	DS

2.5 EXTENDED OPERATORS

Extended operators are those operators which can be derived from basic operators. There are mainly three types of extended operators in Relational Algebra. They are

- **Join**
- **Intersection**
- **Divide**

2.5.1 Join Operation

Join operation is used to combine two or more tables based on the related attributes. It is basically a cross product followed by some more operations like select, project etc. There are mainly two types of join which are further divided as follows:

1. **Inner Join**
   - Natural Join
   - Theta Join
   - Equi Join
2. **Outer Join**
   - Left Outer Join
   - Right Inner Join
   - Full Outer Join

2.5.1.1 Inner Join

Inner join is a type of join in which only those tuples are selected which full fill the required conditions. All those tuples which do not satisfy the required conditions are excluded. Let us now study in detail each type of inner join with example.

Natural Join(⋈)

Natural Join is a join which is performed if there is a common attribute between the relations.

Notation: R1 ⋈ R2 where R1 and R2 are two relations.

Example: We have two tables of Student(S_id, Name, Age, C_id) and Course(C_id, C_name).

Student

S_ID	NAME	AGE	C_ID
1	Andrew	25	11
2	Dhee	30	11
3	Vandhana	31	22

Course

C_ID	C_NAME
11	Fundamentals of C
21	C++

Now, we will perform natural join on both the tables.

S_ID	NAME	AGE	C_ID	C_NAME
1	Andrew	25	11	Fundamentals of C
2	Dhee	30	11	Fundamentals of C

Theta join: is a join which combines the tuples from different relations according to the given theta condition. The join condition in theta join is denoted by **theta (θ)** symbol. This join uses all kind of comparison operator.

Notation: $R_1 \bowtie R_2$

where R_1 and R_2 are relations such that they don't have any common attribute.

Example: We have two tables of Student1(S_id, Name, Std, Age) and Course1 (Class, C_name)

Student 1.

S_ID	NAME	STD	AGE
1	Andrew	10	25
2	Dhee	9	30
3	Vandhana	8	31

Course

C_ID	C_NAME
8	Fundamentals of C
9	C++

Now, we will perform theta join on both the tables

Student* 1** $(\bowtie_{(Student1.std=Course1.class)})$ ***Course 1

S_ID	NAME	AGE	CLASS	C_NAME
1	Andrew	25	11	Fundamentals of C
2	Dhee	30	11	Fundamentals of C

Equi Join: Equi Join is a type of theta join where we use only the equality operator, unlike theta join where we can use any operator. The above example which we gave in the theta join is also an example of equi-join.

2.5.1.2 Outer Join

In Inner Join, we matched rows are returned and unmatched rows are not returned. But, in outer join, we include those tuples which meet the given condition along with that, we also add those tuples which do not meet the required condition. The result also includes the tuples from the left and right tables which do not satisfy the conditions. Based on the tuples that are added from left, right or both the tables, the outer join is further divided into three types. We will now study about its types with the help of examples.

Left Outer Join (⋈): Left Outer Join is a type of join in which all the tuples from left relation are included and only those tuples from right relation are included which have a common value in the common attribute on which the join is being performed.

Notation: R1 ⋈ R2 where R1 and R2 are relations.

Example: We use the above two tables Student (S_id, Name, Age, C_id) and Course(C_id, C_name). Now, we will perform left outer join on both the tables i.e

S_ID	NAME	AGE	C_ID	C_NAME
1	Andrew	25	11	Fundamentals of C
2	Dhee	30	11	Fundamentals of C
3	Vandhana	31	22	---

We can see that the new resulting table has all the tuples from the **Student** table but it doesn't have that tuple from the **course** table whose attributes values was not matching. Also, it fills the table with the null value for those columns whose value is not defined.

Right Outer Join(⋈): Right Outer Join is a type of join in which all the tuples from right relation are included and only those tuples from left relation are included which have a common value in the common attribute on which the right join is being performed.

Notation: R1 ⋈ R2 where R1 and R2 are relations.

Example: We use the above two tables Student(S_id, Name, Age, C_id) and Course(C_id, C_name). Now, we will perform right outer join on both the tables i.e

Student ⋈ Course

S_ID	NAME	AGE	C_ID	C_NAME
1	Andrew	25	11	Fundamentals of C
2	Dhee	30	11	Fundamentals of C
---	----	----	21	C++

We can see that the new resulting table has all the tuples from the **Course** table but it doesn't have that tuple from the **Student** table whose attributes values was not matching. Also, it fills the table with the null value for those columns whose value is not defined.

Full Outer Join(⋈): Full Outer Join is a type of join in which all the tuples from the left and right relation which are having the same value on the common attribute. Also, they will have all the remaining tuples which are not common on in both the relations.

Notation: R1 ⋈ R2 where R1 and R2 are relations.

Example: We use the above two tables Student(S_id, Name, Age, C_id) and Course(C_id, C_name). Now, we will perform full outer join on both the tables i.e

S_ID	NAME	AGE	C_ID	C_NAME
1	Andrew	25	11	Fundamentals of C
2	Dhee	30	11	Fundamentals of C
3	Vandhana	31	22	---
----	----	----	21	C++

We can see that the new resulting table has all the tuples from the **Course** table as well as the **Student** table. Also, it fills the table with the null value for those columns whose value is not defined.

2.5.2 Divide Operation

Division operator A÷B can be applied if and only if:

- Attributes of B is proper subset of Attributes of A.
- The relation returned by division operator will have attributes = (All attributes of A – All Attributes of B)
- The relation returned by division operator will return those tuples from relation A which are associated to every B's tuple.

Example:

Consider the following Student and Student1 relation. The schema are

Student(ID, Name, Subject, Age) and Student1(Subject, Age)

ID	Name	Subject	Age
100	Ashish	Maths	19
200	Rahul	Science	20
300	Naina	Physics	20
400	Sameer	Chemistry	21

Subject	Age
Maths	19
Science	20
Physics	20
Chemistry	21

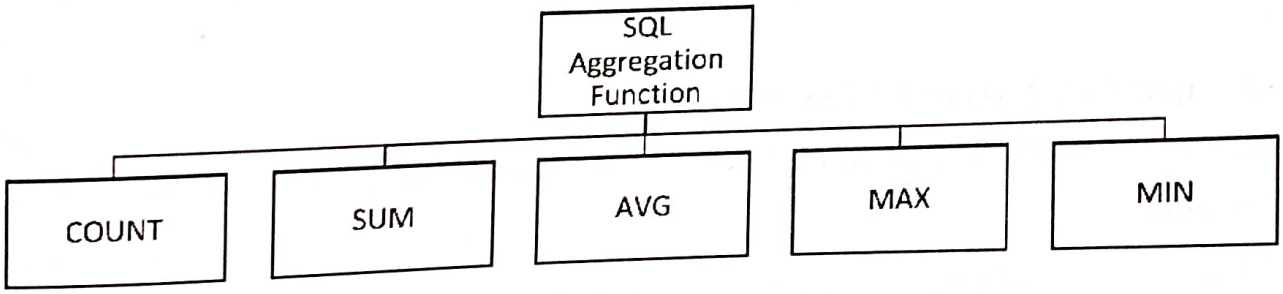
The resultant of A÷B is

ID	Name
100	Ashish
200	Rahul
300	Naina
400	Sameer

2.6 SQL AGGREGATE FUNCTIONS

- SQL aggregation function is used to perform the calculations on multiple rows of a single column of a table. It returns a single value.
- It is also used to summarize the data.

Types of SQL Aggregation Function



2.1.6 COUNT Function

- COUNT function is used to Count the number of rows in a database table. It can work on both numeric and non-numeric data types.
- COUNT function uses the COUNT(*) that returns the count of all the rows in a specified table. COUNT(*) considers duplicate and Null.

Syntax

COUNT(*) or

COUNT ([ALL|DISTINCT] expression)

Example:

Consider the PRODUCT table

PRODUCT	COMPANY	QTY	RATE	COST
Keyboard	ABC	2	700	1400
Mouse	ABC	4	600	2400
Processor	Intel	2	5600	11200
Printer	HP	2	3000	6000

Example: COUNT()

SELECT COUNT(*) FROM PRODUCT;

Output: 10

Example: COUNT with WHERE

SELECT COUNT(*) FROM PRODUCT WHERE RATE>=1000;

Output:

2

Example: COUNT() with DISTINCT

SELECT COUNT(DISTINCT COMPANY) FROM PRODUCT;

Output:

3

Example: COUNT() with GROUP BY

SELECT COMPANY, COUNT(*) FROM PRODUCT GROUP BY COMPANY;

Output:

ABC 2

Intel 1

HP 1

Example: COUNT() with HAVING

SELECT COMPANY, COUNT(*) FROM PRODUCT

GROUP BY COMPANY HAVING COUNT(*)>1;

Output:

ABC 2

2.6.2 SUM Function

Sum function is used to calculate the sum of all selected columns. It works on numeric fields only.

Syntax

SUM()

or

SUM([ALL|DISTINCT] expression)

Example: SUM()

SELECT SUM(COST) FROM PRODUCT;

Output:

21000

Example: SUM() with WHERE

SELECT SUM(COST) FROM PRODUCT WHERE QTY>3;

Output:

2400

Example: SUM() with GROUP BY

SELECT SUM(COST) FROM PRODUCT WHERE QTY>1 GROUP BY COMPANY;

Output:

ABC	3800
Intel	11200
HP	6000

Example: SUM() with HAVING

SELECT COMPANY, SUM(COST) FROM PRODUCT GROUP BY COMPANY
HAVING SUM(COST)>=5000;

Output:

Intel	11200
HP	6000

2.6.3 AVG Function

The AVG function is used to calculate the average value of the numeric type. AVG function returns the average of all non-Null values.

Syntax

AVG()

or

AVG([ALL|DISTINCT] expression)

Example:

SELECT AVG(COST) FROM PRODUCT;

Output:

5250

2.6.4 MAX Function

MAX function is used to find the maximum value of a certain column. This function determines the largest value of all selected values of a column.

Syntax

MAX()

or

MAX([ALL|DISTINCT] expression)

Example:

SELECT MAX(RATE) FROM PRODUCT;

Output:

5600

2.6.5 MIN Function

MIN function is used to find the minimum value of a certain column. This function determines the smallest value of all selected values of a column.

Syntax

ABC	2	700	1400
ABC	4	600	2400
Intel	2	5600	11200
HP	2	3000	6000

MIN()

or

MIN([ALL|DISTINCT] expression)

Example: SELECT MIN(RATE) FROM PRODUCT;

Output: 600

2.7 NESTED SUB QUERIES

A nested query is a quey that has another query embedded within it. The embedded query is called a subquery. Consider the following tables as example:

MovieStar(name: string, address: string, gender: char, birthdate: date)

StarsIn(movieTitle: string, movieYear: string, starName: string)

Consider the following SELECT-FROM-WHERE statement.

SELECT movieTitle FROM StarsIn, MovieStar WHERE starName = name AND birthdate = 1960

Its translation is as follows:

$$\pi_{movieTitle}\ \sigma_{starName\ =\ name\ \wedge\ birthdate\ =\ 1960)}\ (StarsIn \times MovieStar)$$

The following query contains a subquery that refers to the starName attribute of the outer relation StarsIn.

```
SELECT movieTitle FROM StarsIn
WHERE EXISTS (SELECT name FROM MovieStar
WHERE birthdate = 1960 AND name = starName)
```

To translate a SELECT– FROM–WHERE statement that is used as a subquery we must make the following modification.

- We must add all context relations to the cartesian product of the relations in the FROM list
- We must add all parameters as attributes to the projection π.

$$\pi_{movieTitle,movieYear,starName,name}\ \sigma_{birthdate=1960\ \square name=starName}\ (StarsIn \times MovieStar)$$

2.8 VIEWS IN SQL

Views in SQL are considered as a virtual table. A view also contains rows and columns. To create the view, we can select the fields from one or more tables present in the database. A view can either have specific rows based on certain condition or all the rows of a table.

Sample table:

Student_Detail

STU_ID	NAME	ADDRESS
1	Stephan	Delhi
2	Kathrin	Noida
3	David	Ghaziabad
4	Alina	Gurugram

Student_Marks

STU_ID	NAME	MARKS	AGE
1	Stephan	97	19
2	Kathrin	86	21
3	David	74	18
4	Alina	90	20
5	John	96	18

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

Syntax:

CREATE VIEW view_name AS

SELECT column1, column2.....

FROM table_name

WHERE condition;

Creating View from a single table

In this example, we create a View named DetailsView from the table Student_Detail.

CREATE VIEW DetailsView AS
SELECT NAME, ADDRESS
FROM Student_Details
WHERE STU_ID < 4;

Just like table query, we can query the view to view the data.

SELECT * FROM DetailsView;

Output:

NAME	ADDRESS
Stephan	Delhi
Kathrin	Noida
David	Ghaziabad

Creating View from multiple tables: View from multiple tables can be created by simply include multiple tables in the SELECT statement.

In the given example, a view is created named MarksView from two tables Student_Detail and Student_Marks.

CREATE VIEW MarksView AS

SELECT Student_Detail.NAME, Student_Detail.ADDRESS, Student_Marks.MARKS

FROM Student_Detail, Student_Mark

WHERE Student_Detail.NAME = Student_Marks.NAME;

To display data of View MarksView:

SELECT * FROM MarksView;

NAME	ADDRESS	MARKS
Stephan	Delhi	97
Kathrin	Noida	86
David	Ghaziabad	74
Alina	Gurugram	90

Deleting View

A view can be deleted using the Drop View statement.

Syntax

DROP VIEW view_name;

Example: If we want to delete the View **MarksView**, we can do this as:

DROP VIEW MarksView;

Review Questions:

1. What is relational algebra?
2. Define Domain, Attribute, Tuple?
3. Explain the characteristics of relation?
4. Explain unary operators in relational algebra?
5. Explain projection operator with anexample?
6. Explain binary operators in relational algebra?
7. Explain set theory operators in relational algebra?
8. Explain cross product?
9. What is join operator? Explain its types with an example?
10. Explain SQL aggregate function with example?
11. Explain nested queries with example?
12. Define view? Explain with an example.
13. Explain inner join with an example?
14. Explain outer join with an example?

PL/SQL: is a block structured lang that enables developers to combine the power of SQL with procedural statements. All the statements of a block are passed to oracle engine all at once which increase processing speed & decreases the traffic.

3 PL/SQL

3.1 INTRODUCTION

The PL/SQL programming language was developed by Oracle Corporation in the late 1980s as procedu extension language for SQL and the Oracle relational database. Following are the facts about PL/SQ

- PL/SQL is a completely portable, high-performance transaction-processing language.
- PL/SQL provides a built-in, interpreted and OS independent programming environment.
- PL/SQL can also directly be called from the command-line **SQL*Plus interface**.
- Direct call can also be made from external programming language calls to database.
- PL/SQL's general syntax is based on that of ADA and Pascal programming language.

3.2 FEATURES OF PL/SQL

PL/SQL has the following features:

- PL/SQL is tightly integrated with SQL.
- It offers extensive error checking.
- It offers numerous data types.
- It offers a variety of programming structures.
- It supports structured programming through functions and procedures.
- It supports object-oriented programming.
- It supports the development of web applications and server pages.

3.3 STRUCTURE OF PL/SQL BLOCK

PL/SQL extends SQL by adding constructs found in procedural languages, resulting in a structura language that is more powerful than SQL. The basic unit in PL/SQL is a block. All PL/SQL programs are made up of blocks, which can be nested within each other.

A block has the following structure:

```
DECLARE
   declaration statements;
BEGIN
   executable statements
EXCEPTIONS
    exception handling statements
END;
```

- Declare section starts with **DECLARE** keyword in which variables, constants, records as cursors can be declared which stores data temporarily. It basically consists definition of PL/SQL identifiers. This part of the code is optional.
- Execution section starts with **BEGIN** and ends with **END** keyword. This is a mandatory section and here the program logic is written to perform any task like loops and conditional statements. It supports all **DML** commands, **DDL** commands and SQL*PLUS built-in functions as well.
- Exception section starts with **EXCEPTION** keyword. This section is optional which contains statements that are executed when a run-time error occurs. Any exceptions can be handled in this section.

3.4 PL/SQL IDENTIFIERS

There are several PL/SQL identifiers such as variables, constants, procedures, cursors, triggers etc.

3.4.1 Variables

Variables in PL/SQL must be declared prior to its use. They should have a valid name and data type as well.

Syntax

variable_name datatype [NOT NULL := value];

Example

var1 INTEGER;

Initializing variables: The variables can also be initialised just like in other programming languages. Let us see an example for the same.

Example

var1 INTEGER := 2 ;

Displaying Output:

The outputs are displayed by using DBMS_OUTPUT which is a built-in package that enables the user to display output, debugging information, and send messages from PL/SQL blocks, subprograms, packages, and triggers.

Example:

```
SQL> SET SERVEROUTPUT ON;
SQL> DECLARE
  var varchar2(40) := ‹Welcome to PL/SQL' ;
 BEGIN
  dbms_output.put_line(var);
 END;
/
```

Output:

```
Welcome to PL/SQL
```

PL/SQL procedure successfully completed.

3.4.2 Constants

A constant declaration specifies its name, data type, and value, and allocates storage for it. The declaratio can also impose the **NOT NULL constraint**.

Declaring a Constant: A constant is declared using the **CONSTANT** keyword. It requires a initial value and does not allow that value to be changed. For example

```
PI CONSTANT NUMBER := 3.141592654;
DECLARE
  -- constant declaration
  pi constant number := 3.141592654;
  -- other declarations
  radius number(5,2);
  dia number(5,2);
  circumference number(7, 2);
  area number (10, 2);
BEGIN
  -- processing
  radius := 9.5;
  dia := radius * 2;
  circumference := 2.0 * pi * radius;
  area := pi * radius * radius;
  -- output
  dbms_output.put_line('Radius: '|| radius);
  dbms_output.put_line('Diameter: '|| dia);
  dbms_output.put_line('Circumference: '|| circumference);
  dbms_output.put_line('Area: '|| area);
END;
/
```

When the above code is executed at the SQL prompt, it produces the following result –

Output:

```
Diameter: 19
Circumference: 59.69
Area: 283.53
```

Pl/SQL procedure successfully completed.

3.4.3 Literals

A literal is an explicit numeric, character, string, or Boolean value not represented by an identifier. For example, TRUE, 786, NULL, 'Javascript' are all literals of type Boolean, number, or string. PL/SQL, literals are case-sensitive. PL/SQL supports the following kinds of literals –

- Numeric Literals - 6. 6667 0.0 -12.0 3.14159 +7800.00
- Character Literals - 'A' '%' '9' ' ' 'z' '('
- String Literals - 'Hello, world!'
- BOOLEAN Literals - TRUE, FALSE, and NULL
- Date and Time Literals - DATE '1978-12-25';

3.5 COMMENTS

Like in many other programming languages, in PL/SQL also, comments can be put within the code which has no effect in the code. There are two syntaxes to create comments in PL/SQL :

- **Single Line Comment:** To create a single line comment , the symbol – – is used.
- **Multi Line Comment:** To create comments that span over several lines, the symbol /* and */ is used.

3.6 OPERATORS

An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulation. PL/SQL language is rich in built-in operators and provides the following types of operators:

- Arithmetic operators
- Relational operators
- Comparison operators
- Logical operators
- String operators

Here, we will understand the arithmetic, relational, comparison and logical operators one by one.

Arithmetic Operators: Following table shows all the arithmetic operators supported by PL/SQL. Let us assume **variable A** holds 10 and **variable B** holds 5, then

Operator	Description	Example
+	Adds two operands	A + B will give 15
-	Subtracts second operand from the first	A - B will give 5
*	Multiplies both operands	A * B will give 50
/	Divides numerator by de-numerator	A / B will give 2
**	Exponentiation operator, raises one operand to the power of other	A ** B will give 100000

Relational Operators: Relational operators compare two expressions or values and return a Boolean result. Following table shows all the relational operators supported by PL/SQL. Let us assume **variable A** holds 10 and **variable B** holds 20, then

Operator	Description	Example
=	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A = B) is not true.
!= <> ~=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true

Comparison Operators: Comparison operators are used for comparing one expression to another. The result is always either **TRUE, FALSE** or **NULL**.

Operator	Description	Example
LIKE	The LIKE operator compares a character, string, or CLOB value to a pattern and returns TRUE if the value matches the pattern and FALSE if it does not.VV	If 'Zara Ali' like 'Z% A_i' returns a Boolean true, whereas, 'Nuha Ali' like 'Z% A_i' returns a Boolean false.
BETWEEN	The BETWEEN operator tests whether a value lies in a specified range. x BETWEEN a AND b means that x >= a and x <= b.VVVV	If x = 10 then, x between 5 and 20 returns true, x between 5 and 10 returns true, but x between 11 and 20 returns false.
IN	The IN operator tests set membership. x IN (set) means that x is equal to any member of set.	If x = 'm' then, x in ('a', 'b', 'c') returns Boolean false but x in ('m', 'n', 'o') returns Boolean true.
IS NULL	The IS NULL operator returns the BOOLEAN value TRUE if its operand is NULL or FALSE if it is not NULL. Comparisons involving NULL values always yield NULL.	If x = 'm', then 'x is null' returns Boolean false.

Logical Operators: Following table shows the Logical operators supported by PL/SQL. All these operators work on Boolean operands and produce Boolean results. Let us assume **variable A** holds true and **variable B** holds false, then

Operator	Description	Example
And	Called the logical AND operator. If both the operands are true then condition becomes true.	(A and B) is false.
Or	Called the logical OR Operator. If any of the two operands is true then condition becomes true.	(A or B) is true.
Not	Called the logical NOT Operator. Used to reverse the logical state of its operand. If a condition is true then Logical NOT operator will make it false.	not (A and B) is true.

PL/SQL Operator Precedence: Operator precedence determines the grouping of terms in an expression. This affects how an expression is evaluated. Certain operators have higher precedence than others. For example, the multiplication operator has higher precedence than the addition operator.

For example, **x = 7 + 3 * 2**; here, **x** is assigned **13**, not 20 because operator * has higher precedence than +, so it first gets multiplied with **3*2** and then adds into **7**.

Here, operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

The precedence of operators goes as follows: =, <, >, <=, >=, <>, !=, ~=, ^=, IS NULL, LIKE, BETWEEN, IN.

Operator	Operation
**	Exponentiation
+, -	identity, negation
*, /	multiplication, division
+, -, ||	addition, subtraction, concatenation
Comparison	
NOT	logical negation
AND	Conjunction
OR	Inclusion

Example 1: Taking input from user

Just like in other programming languages, in PL/SQL also, we can take input from the user and store it in a variable. Let us see an example to show how to take input from users in PL/SQL:

```
SQL> SET SERVEROUTPUT ON;
 SQL> DECLARE
    -- taking input for variable a
    a number := &a;
        -- taking input for variable b
    b varchar2(30) := &b;
  BEGIN
   null;
  END;
 /
```

Output:

```
Enter value for a: 24
old   2: a number := &a;
new   2: a number := 24;
Enter value for b: 'Welcome'
old   3: b varchar2(30) := &b;
new   3: b varchar2(30) := 'Welcome';
```

PL/SQL procedure successfully completed.

```
DECLARE
  a integer := 10;
  b integer := 20;
  c integer;
  f real;
BEGIN
  c := a + b;
  dbms_output.put_line('Value of c: ' || c);
  f := 70.0/3.0;
  dbms_output.put_line('Value of f: ' || f);
END;
/
```

When the above code is executed, it produces the following result –

Value of c: 30

Value of f: 23.333333333333333

PL/SQL procedure successfully completed.

Example 2: Assigning SQL Query Results to PL/SQL Variables

You can use the **SELECT INTO** statement of SQL to assign values to PL/SQL variables. For each item in the **SELECT list**, there must be a corresponding, type-compatible variable in the **INTO list**. The following example illustrates the concept. Let us create a table named CUSTOMER

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

Table Created
```

Let us now insert some values in the table

```
INSERT INTO CUSTOMER(ID,NAME,AGE,ADDRESS,SALARY)
VALUES (1, 'Varun', 23, 'Kerala', 30000.00 );

INSERT INTO CUSTOMER(ID,NAME,AGE,ADDRESS,SALARY)
VALUES (2, 'Karun', 21, 'Bangalore', 28000.00 );

INSERT INTO CUSTOMER(ID,NAME,AGE,ADDRESS,SALARY)
VALUES (3, 'Vijaya', 22, 'Delhi', 31000.00 );

INSERT INTO CUSTOMER(ID,NAME,AGE,ADDRESS,SALARY)
VALUES (4, 'Hema', 24, 'Varanasi', 33000.00 );

INSERT INTO CUSTOMER(ID,NAME,AGE,ADDRESS,SALARY)
VALUES (5, 'Abinaya', 21, 'Mumbai', 29000.00 );

INSERT INTO CUSTOMER(ID,NAME,AGE,ADDRESS,SALARY)
VALUES (6, 'Rithivik', 22, 'Punjab', 30000.00 );
```

The following program assigns values from the above table to PL/SQL variables using the **SELECT INTO clause** of SQL

```
DECLARE
  c_id customers.id%type := 1;
  c_name  customers.name%type;
  c_addr customers.address%type;
  c_sal  customers.salary%type;
BEGIN
  SELECT name, address, salary INTO c_name, c_addr, c_sal
  FROM customer
  WHERE id = c_id;
  dbms_output.put_line
  ('Customer ' ||c_name || ' from ' || c_addr || ' earns ' || c_sal);
END;
/
```

When the above code is executed, it produces the following result –

Output

```
Customer Varun from Kerala earns 30000
```

PL/SQL procedure completed successfully

3.7 CONDITIONAL STATEMENTS

PL/SQL programming language provides following types of decision-making statements.

IF-THEN statement

Syntax

```
IF condition
THEN
Statements;
END IF;
```

The **IF statement** associates a condition with a sequence of statements enclosed by the keywords **THEN** and **END IF**. If the condition is true, the statements get executed and if the condition is false or NULL then the IF statement does nothing.

IF-THEN-ELSE statement

Syntax

```
IF condition
THEN
  {...statements to execute when condition is TRUE...}
ELSE
  {...statements to execute when condition is FALSE...}
END IF;
```

IF statement adds the keyword **ELSE** followed by an alternative sequence of statement. If the condition is false or NULL, then only the alternative sequence of statements get executed. It ensures that either of the sequence of statements is executed.

IF-THEN-ELSIF-ELSE statement

Syntax

```
IF condition1
THEN
  {...statements to execute when condition1 is TRUE...}
  ELSIF condition2
THEN
  {...statements to execute when condition2 is TRUE...}
ELSE
```

{...statements to execute when both condition1 and condition2 are FALSE...}

END IF;

It allows you to choose between several alternatives.

Example

1. **Write a PL/SQL program to illustrate if-then-else statement.**

```
DECLARE
   a number(3) := 500;
BEGIN
   IF( a < 20 ) THEN
       dbms_output.put_line(‹a is less than 20 ‹ );
  ELSE
    dbms_output.put_line(‹a is not less than 20 ‹ );
  END IF;
  dbms_output.put_line(‹value of a is : ‹ || a);
END;
```

Output

```
a is not less than 20
value of a is : 500
PL/SQL procedure successfully completed.
```

Case statement: Like the IF statement, the **CASE statement** selects one sequence of statements to execute. However, to select the sequence, the CASE statement uses a selector rather than multiple Boolean expressions. A selector is an expression whose value is used to select one of several alternatives.

Syntax

CASE [expression]

WHEN condition_1 **THEN** result_1

WHEN condition_2 **THEN** result_2

...

WHEN condition_n **THEN** result_n

ELSE result

END

Example

1. **Write a program to illustrate CASE statement**

```
DECLARE
  grade char(1) := 'A';
BEGIN
  CASE grade
    when 'A' then dbms_output.put_line(‹Excellent›);
    when 'B' then dbms_output.put_line(‹Very good›);
    when 'C' then dbms_output.put_line(‹Good›);
    when 'D' then dbms_output.put_line(‹Average›);
    when 'F' then dbms_output.put_line(‹Passed with Grace›);
    else dbms_output.put_line(‹Failed›);
  END CASE;
END;
```

Output

```
Excellent
PL/SQL procedure successfully completed.
```

3.8 ITERATIVE STATEMENTS

A loop statement allows us to execute a statement or group of statements multiple times. PL/SQL provides the following types of loops to handle the looping requirements.

Basic LOOP/Exit LOOP: In this loop structure, sequence of statements is enclosed between the LOOP and the END LOOP statements. At each iteration, the sequence of statements is executed and then control resumes at the top of the loop.

Syntax for Basic LOOP

```
  LOOP
  Sequence of statements;
 END LOOP;
```

Syntax for Exit LOOP

```
  LOOP
    statements;
EXIT;
{or EXIT WHEN condition;}
END LOOP;
```

Example

2. **Write a PL/SQL program to illustrate the basic/exit loop**

```
DECLARE
 i NUMBER := 1;
BEGIN
LOOP
EXIT WHEN i>5;
DBMS_OUTPUT.PUT_LINE(i);
i := i+1;
END LOOP;
END;
```

Output

```
1
2
3
4
5
```

WHILE LOOP

Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.

```
WHILE <condition>
 LOOP statements;
END LOOP;
```

Example

1. **Write a PL/SQL program to illustrate the while loop**

```
DECLARE
i INTEGER := 1;
sum INTEGER := 0;
BEGIN
WHILE i <= 10 LOOP
DBMS_OUTPUT.PUT_LINE(i);
sum:=sum+I;
i := i+1;
END LOOP;
dbms_output.put_line('Sum is ' || sum);
END;
```

Output

```
1
2
3
4
5
Sum is 15
```

FOR LOOP

Execute a sequence of statements multiple times and abbreviates the code that manages the loop variable.

Syntax

FOR counter IN initial_value .. final_value LOOP

LOOP statements;

END LOOP;

Where **initial_value** is a Start integer value and **final_value** is an End integer value

Example

1. Write a PL/SQL program to illustrate the for loop

```
DECLARE
n NUMBER;
BEGIN
n:=7;
FOR i IN 1..10
LOOP
DBMS_OUTPUT.PUT_LINE (n||'*'||i||'='||i*n);
END LOOP;
END;
```

Output

```
7*1=7
7*2=14
7*3=21
7*4=28
7*5=35
7*6=42
7*7=49
7*8=56
7*9=63
7*10=70
```

3.9 PL/SQL FUNCTION

PL/SQL function is a named block that returns a value. A PL/SQL function is also known as a subroutine or a subprogram. To create a PL/SQL function, use the following syntax:

```
CREATE [OR REPLACE] FUNCTION function_name [parameters]
[(parameter_name [IN | OUT | IN OUT] type [, ...])]
RETURN return_datatype
{IS | AS}
BEGIN
  < function_body >
END [function_name];
```

- **Function_name:** specifies the name of the function.
- **[OR REPLACE]** option allows modifying an existing function.
- The **optional parameter list** contains name, mode and types of the parameters.
- **IN** represents that value will be passed from outside and OUT represents that this parameter will be used to return a value outside of the procedure.
- The function must contain a return statement.
- RETURN clause specifies that data type you are going to return from the function.
- Function_body contains the executable part.
- The AS keyword is used instead of the IS keyword for creating a standalone function.

Example

The following function add two numbers:

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

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

The following program is used to call the above Sum function

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

Output

Addition is: 33

Example 2

The following program find the largest of two numbers using function

```
DECLARE
  a number;
  b number;
  c number;
FUNCTION Largest(x IN number, y IN number)
   RETURN number
   IS
   z number;
  BEGIN
     IF x > y THEN
         z:= x;
  ELSE
        z:= y;
  END IF;
       RETURN z;
 END;
BEGIN
  a:= 23;
  b:= 45;
  c := Largest(a, b);
  dbms_output.put_line(‹ Maximum of (23,45): ‹ || c);
END;
/
```

Output

Maximum of (23,45): 45

3.10 PL/SQL PROCEDURE

A PL/SQL procedure is a named block that does a specific task. PL/SQL procedure allows you to encapsulate complex business logic and reuse it in both database layer and application layer.

The following illustrates the PL/SQL procedure's syntax:

```
CREATE [OR REPLACE] PROCEDURE procedure_name  [ ( parameter_1 [IN] [OUT]
data_type,
  parameter_2 [IN] [OUT] data_type,
  parameter_N [IN] [OUT] data_type)]
IS
   [declaration_section]
BEGIN
   executable_section
[EXCEPTION
   exception_section]
END [procedure_name];
```

- An **IN** parameter is a read-only parameter. If the procedure tries to change the value of the IN parameters, the compiler will issue an error message.
- An **OUT** parameter is a write-only parameter. The OUT parameters are used to return values back to the calling program. An OUT parameter is initialized to a default value of its type.
- An **IN OUT** parameter is read and write parameter. It means the procedure reads the value from an IN OUT parameter, change its value and return it back to the calling program.

Example

Consider a **user** table is created using the following query

create table user(id number(10) primary key,name varchar2(100));

Let us insert a record using the following procedure.

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

Output:

```
Procedure Created.
```

PL/SQL program to call procedure

To call above created procedure use the following PL/SQL program

```
BEGIN
  insertuser(101,›Rahul›);
  dbms_output.put_line(‹record inserted successfully›);
END;
/
```

Now **user** table contain one record

ID	NAME
101	Rahul

3.11 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.**

One 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.

The following table specifies the status of the cursor with each of its attribute.

Attribute	Description
%FOUND	Its return value is TRUE if DML statements like INSERT, DELETE and UPDATE affect at least one row or more rows or a SELECT INTO statement returned one or more rows. Otherwise it returns FALSE.

%NOTFOUND	Its return value is TRUE if DML statements like INSERT, DELETE and UPDATE affect no row, or a SELECT INTO statement return no rows. Otherwise it returns FALSE. It is a just opposite of %FOUND.
%ISOPEN	It always returns FALSE for implicit cursors, because the SQL cursor is automatically closed after executing its associated SQL statements.
%ROWCOUNT	It returns the number of rows affected by DML statements like INSERT, DELETE, and UPDATE or returned by a SELECT INTO statement.

Consider the following **Customer** table:

ID	NAME	AGE	ADDRESS	SALARY
1	Varun	23	Kerala	30000
2	Karun	21	Bangalore	28000
3	Vijaya	22	Delhi	31000
4	Hema	24	Varanasi	33000
5	Abinaya	21	Mumbai	29000
6	Rithivik	22	Punjab	30000

Let's execute the following program to update the table and increase salary of each customer by 5000. Here, SQL%ROWCOUNT attribute is used to determine the number of rows affected

```
DECLARE
  total_rows number(2);
BEGIN
  UPDATE  customer
  SET salary = salary + 5000;
  IF sql%notfound THEN
    dbms_output.put_line(‹no customers updated›);
  ELSIF sql%found THEN
    total_rows := sql%rowcount;
    dbms_output.put_line( total_rows || ‹ customers updated ‹);
  END IF;
END;
/
```

Output:

6 customers updated

PL/SQL procedure successfully completed.

After execution the **customer** table looks as follows:

ID	NAME	AGE	ADDRESS	SALARY
1	Varun	23	Kerala	35000
2	Karun	21	Bangalore	33000
3	Vijaya	22	Delhi	36000
4	Hema	24	Varanasi	38000
5	Abinaya	21	Mumbai	34000
6	Rithivik	22	Punjab	35000

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.

The syntax for creating an explicit cursor is

CURSOR cursor_name **IS** select_statement;

Working with 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

Declaring the Cursor: Declaring the cursor defines the cursor with a name and the associated SELECT statement. For example

```
CURSOR c_customer IS
  SELECT id, name, address FROM customer;
```

Opening the Cursor: Opening the cursor allocates the memory for the cursor and makes it ready for fetching the rows returned by the SQL statement into it. For example, we will open the above defined cursor as follows:

```
OPEN c_customer;
```

Fetching the Cursor: Fetching the cursor involves accessing one row at a time. For example, we will fetch rows from the above-opened cursor as follows –

```
FETCH c_customer INTO c_id, c_name, c_addr;
```

Closing the Cursor: Closing the cursor means releasing the allocated memory. For example, we will close the above-opened cursor as follows –

```
CLOSE c_customer;
```

Example

Following is a complete example to illustrate the concepts of explicit cursors &minua;

```
DECLARE
  c_id customer.id%type;
  c_name customer.name%type;
  c_addr customer.address%type;
  CURSOR c_customer is
    SELECT id, name, address FROM customer;
BEGIN
  OPEN c_customer;
  LOOP
  FETCH c_customer into c_id, c_name, c_addr;
    EXIT WHEN c_customer%notfound;
    dbms_output.put_line(c_id || ' ' || c_name || ' ' || c_addr);
  END LOOP;
  CLOSE c_customer;
END;
/
```

When the above code is executed at the SQL prompt, it produces the following result –

1	Varun	Kerala
2	Karun	Bangalore
3	Vijaya	Delhi
4	Hema	Varanasi
5	Abinaya	Mumbai
6	Rithivik	Punjab

3.12 PL/SQL EXCEPTION

An error occurs during the program execution is called Exception in PL/SQL. PL/SQL facilitates programmers to catch such conditions using exception block in the program and an appropriate action is taken against the error condition.

There are two type of exceptions:

- System-defined Exceptions- which are executed when any database rule is violated by the programs.
- User-defined Exceptions - A user-defined exception can be raised explicitly, using either a RAISE statement or the procedure DBMS_STANDARD.RAISE_APPLICATION_ERROR.

Syntax

```
DECLARE
  <declarations section>
BEGIN
  <executable command(s)>
EXCEPTION
  <exception handling goes here >
  WHEN exception1 THEN
    exception1-handling-statements
  WHEN exception2  THEN
    exception2-handling-statements
  WHEN exception3 THEN
    exception3-handling-statements
  ........
  WHEN others THEN
    exception3-handling-statements
END;
```

Example

Consider the following customer table:

ID	NAME	AGE	ADDRESS	SALARY
1	Varun	23	Kerala	30000
2	Karun	21	Bangalore	28000
3	Vijaya	22	Delhi	31000
4	Hema	24	Varanasi	33000
5	Abinaya	21	Mumbai	29000
6	Rithivik	22	Punjab	30000

The following program illustrates the PL/SQL exception

```
DECLARE
c_id customers.id%type := 8;
c_name  customers.name%type;
c_addr customers.address%type;
BEGIN
SELECT  name, address INTO  c_name, c_addr
FROM customers
WHERE id = c_id;
DBMS_OUTPUT.PUT_LINE ('Name: '||  c_name);
DBMS_OUTPUT.PUT_LINE ('Address: ' || c_addr);
EXCEPTION
WHEN no_data_found THEN
dbms_output.put_line('No such customer!');
WHEN others THEN
dbms_output.put_line('Error!');
END;
/
```

After the execution of above code at SQL Prompt, it produces the following result:

No such customer!

PL/SQL procedure successfully completed.

The above program should show the name and address of a customer as result whose ID is given. But there is no customer with ID value 8 in our database, so the program raises the run-time exception NO_DATA_FOUND, which is captured in EXCEPTION block.

If you use the id defined in the above table (i.e. 1 to 6), you will get a certain result. For example: here, we are using the id 2.

```
DECLARE
c_id customers.id%type := 2;
c_name  customers.name%type;
c_addr customers.address%type;
BEGIN
SELECT  name, address INTO  c_name, c_addr
FROM customers
WHERE id = c_id;
DBMS_OUTPUT.PUT_LINE ('Name: '|| c_name);
DBMS_OUTPUT.PUT_LINE ('Address: ' || c_addr);
EXCEPTION
WHEN no_data_found THEN
dbms_output.put_line('No such customer!');
WHEN others THEN
dbms_output.put_line('Error!');
END;
/
```

After the execution of above code at SQL prompt, you will get the following result:

Name: Karun

Address: Bangalore

PL/SQL procedure successfully completed.

3.13 TRIGGERS

Triggers are stored programs, which are automatically executed or fired when some event occurs Triggers are 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).
- A database operation (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN).

Triggers could be defined on the table, view, schema, or database with which the event is associated.

Syntax

```
CREATE [OR REPLACE ] TRIGGER trigger_name
{BEFORE | AFTER | INSTEAD OF }
{INSERT [OR] | UPDATE [OR] | DELETE}
[OF col_name]
ON table_name
[REFERENCING OLD AS o NEW AS n]
[FOR EACH ROW]
WHEN (condition)
DECLARE
  Declaration-statements
BEGIN
  Executable-statements
EXCEPTION
  Exception-handling-statements
END;
```

- CREATE [OR REPLACE] TRIGGER trigger_name: It creates or replaces an existing trigger with the trigger_name.
- {BEFORE | AFTER | INSTEAD OF} : This specifies when the trigger would be executed. The INSTEAD OF clause is used for creating trigger on a view.
- {INSERT [OR] | UPDATE [OR] | DELETE}: This specifies the DML operation.
- [OF col_name]: This specifies the column name that would be updated.
- [ON table_name]: This specifies the name of the table associated with the trigger.
- [REFERENCING OLD AS o NEW AS n]: This allows you to refer new and old values for various DML statements, like INSERT, UPDATE, and DELETE.
- [FOR EACH ROW]: This specifies a row level trigger, i.e., the trigger would be executed for each row being affected. Otherwise the trigger will execute just once when the SQL statement is executed, which is called a table level trigger.
- WHEN (condition): This provides a condition for rows for which the trigger would fire. This clause is valid only for row level triggers.
- Consider the following customer table:

ID	NAME	AGE	ADDRESS	SALARY
1	Varun	23	Kerala	30000
2	Karun	21	Bangalore	28000
3	Vijaya	22	Delhi	31000

Let's take a program to create a row level trigger for the CUSTOMERS table that would fire for INSERT or UPDATE or DELETE operations performed on the CUSTOMERS table. This trigger will display the salary difference between the old values and new values:

```
CREATE OR REPLACE TRIGGER display_salary_changes
BEFORE DELETE OR INSERT OR UPDATE ON customer
FOR EACH ROW
WHEN (NEW.ID > 0)
DECLARE
  sal_diff number;
BEGIN
  sal_diff := :NEW.salary  - :OLD.salary;
  dbms_output.put_line('Old salary: ' || :OLD.salary);
  dbms_output.put_line('New salary: ' || :NEW.salary);
  dbms_output.put_line('Salary difference: ' || sal_diff);
END;
/
```

After the execution of the above code at SQL Prompt, it produces the following result.

Trigger created.

Use the following code to get the old salary, new salary and salary difference after the trigger created.

```
DECLARE
  total_rows number(2);
BEGIN
  UPDATE  customer
  SET salary = salary + 3000;
  IF sql%notfound THEN
    dbms_output.put_line('no customers updated');
  ELSIF sql%found THEN
    total_rows := sql%rowcount;
    dbms_output.put_line( total_rows || ' customers updated ');
  END IF;
END;
/
```

Output

```
Old salary: 30000
New salary: 33000
Salary difference: 3000
Old salary: 28000
New salary: 30000
Salary difference: 3000
Old salary: 31000
New salary: 34000
Salary difference: 3000
3 customers updated
```

Review Questions

1. What is PL/SQL?
2. What are the advantages of PL/SQL?
3. Describe the PL/SQL block?
4. Explain the datatypes in PL/SQL?
5. Explain if then statement in PL/SQL with an example?
6. Explain conditional statement in PL/SQL with an example?
7. Explain CASE statement in PL/SQL with an example?
8. Explain Loop statement in PL/SQL with an example?
9. Explain while loop in PL/SQL with an example?
10. Explain for loop in PL/SQL with an example?
11. Explain function in PL/SQL with an example?
12. Explain procedure in PL/SQL with an example?
13. What is cursor? Explain its types?
14. What is trigger? Explain with an example?
15. Explain exception with an example?

May/June 2014

1) What is SQL?

Ans. **SQL** is a **Structured Query Language** commonly called as database language designed for the retrieval and management of data in a relational database.

2) What is relational algebra?

Ans. Relational algebra is a procedural query language, which takes relation as input and generate relation as output.

3) What is DML? Give example

Ans. Data Manipulation Language (DML) is a set of special commands that allows us to access and manipulate data stored in existing schema objects. Example: Select, Insert, Update, Delete.

4) What is relational schema and relational instance? Give one example.

Ans. A relation schema represents name of the relation with its attributes. e.g.; STUDENT (ROLL_NO, NAME, ADDRESS, PHONE and AGE) is relation schema for STUDENT. If a schema has more than 1 relation, it is called Relational Schema.

The set of tuples of a relation at a particular instance of time is called as relation instance. Table 1 shows the relation instance of STUDENT at a particular time. It can change whenever there is insertion, deletion or updation in the database.

Example: Consider a relation STUDENT with attributes ROLL_NO, NAME, ADDRESS, PHONE and AGE shown in Table 1:

ROLL_NO	NAME	ADDRESS	PHONE	AGE
1	SURYA	KOLAR	9454678998	19
2	RAM	BANGALORE	9890234549	18
3	REKHA	MANGALORE		19
4	ANAND	UDUPI	9667584833	20

5) What is view and its advantage?

Ans. Views in SQL are considered as a virtual table. A view also contains rows and columns. To create the view, we can select the fields from one or more tables present in the database. A view can either have specific rows based on certain condition or all the rows of a table.

Syntax:

CREATE VIEW view_name AS **SELECT** column1, column2..... column n **FROM** table_name **WHERE** condition;

Advantages

- Views don't store data in a physical location.
- The view can be used to hide some of the columns from the table.
- Views can provide Access Restriction, since data insertion, update and deletion is not possible with the view.

6) What is trigger?

Ans. Triggers are stored programs, which are automatically executed or fired when some event occurs. Triggers are 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).
- A database operation (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN).

Triggers could be defined on the table, view, schema, or database with which the event is associated.

Syntax

```
CREATE [OR REPLACE ] TRIGGER trigger_name
{BEFORE | AFTER | INSTEAD OF }
{INSERT [OR] | UPDATE [OR] | DELETE}
[OF col_name]
ON table_name
[REFERENCING OLD AS o NEW AS n]
[FOR EACH ROW]
WHEN (condition)
DECLARE
  Declaration-statements
BEGIN
  Executable-statements
EXCEPTION
  Exception-handling-statements
END;
```

7) Explain selection and projection operation in relational algebra with an example?

Ans. **Selection Operator:** Selection Operator (σ) is a unary operator in relational algebra that performs aselection operation. It selects those rows or tuples from the relation that satisfies the selection condition.

Syntax: σ<selection_condition>(R)

Examples: Consider the following Student relation-

ID	Name	Subject	Age
100	Ashish	Maths	19
200	Rahul	Science	20
300	Naina	Physics	20
400	Sameer	Chemistry	21

Select tuples from a relation "Student" where subject is "database"

σsubject = "Science" (Student)

ID	Name	Subject	Age
200	Rahul	Science	20

Projection: Projection Operator (π) is a unary operator in relational algebra that performs a projection operation. It displays the columns of a relation or table based on the specified attributes.

Syntax: π<attribute list>(R)

Example: Consider the following Student relation-

ID	Name	Subject	Age
100	Ashish	Maths	19
200	Rahul	Science	20
300	Naina	Physics	20
400	Sameer	Chemistry	21

Result for Query **πName, Age(Student)**

Name	Age
Ashish	19
Rahul	20
Naina	20
Sameer	21

8) Explain the following SQL commands with examples

i) Create ii) Alter iii) Select iv) Truncate

Ans. i) CREATE

It is used to create the database or its schema objects. *To create a new table*

Syntax

CREATE TABLE table_name (column_1 DATATYPE,column_2 DATATYPE,column_n DATATYPE);

Example

CREATE TABLE Student (StudID int,LastName varchar(255),FirstName varchar(255),Address varchar(255));

Table Created

The above command will create the table schema

StudID	LastName	FirstName	Address

ii) **ALTER:** It is used to modify the structure of the database objects. To add new column in a table

Syntax

ALTER TABLE table_name ADD (column_1 DATATYPE,column_2 DATATYPE,column_n DATATYPE);

Example: ALTER TABLE Student ADD DOB date;

iii) **SELECT:** It is used to retrieve or fetch data from a database. The SELECT statement cannot manipulate data, it can only access it. Hence, it is known as the Data Query Language. To fetch an entire table

Syntax: SELECT * FROM table_name;

Example: SELECT * FROM Student;

iv) **TRUNCATE:** It is used to remove the whole content of the table along with the deallocation of the space occupied by the data, without affecting the table's structure. To remove data present inside a table

Syntax: TRUNCATE TABLE table_name;

Example: TRUNCATE TABLE Student;

9.a) Explain PL/SQL control statement with an example?

Ans. PL/SQL programming language provides following types of decision-making statements.

IF-THEN statement

Syntax:

IF condition

THEN

Statements;

END IF;

The **IF statement** associates a condition with a sequence of statements enclosed by the keywords **THEN** and **END IF**. If the condition is true, the statements get executed and if the condition is false or NULL then the IF statement does nothing.

IF-THEN-ELSE statement

Syntax:

IF condition
THEN
{...statements **to execute when** condition **is TRUE**...}
ELSE
{...statements **to execute when** condition **is FALSE**...}
END IF;

IF statement adds the keyword **ELSE** followed by an alternative sequence of statement. If the condition is false or NULL, then only the alternative sequence of statements get executed. It ensures that either of the sequence of statements is executed.

IF-THEN-ELSIF-ELSE statement

Syntax:

IF condition1
THEN
{...statements **to execute when** condition1 **is TRUE**...}
ELSIF condition2
THEN
{...statements **to execute when** condition2 **is TRUE**...}
ELSE
{...statements **to execute when** both condition1 and condition2 are **FALSE**...}
END IF;

It allows you to choose between several alternatives.

Example

```
DECLARE
  a number(3) := 500;
BEGIN
   IF( a < 20 ) THEN
       dbms_output.put_line('a is less than 20 ' );
  ELSE
    dbms_output.put_line('a is not less than 20 ' );
  END IF;
  dbms_output.put_line('value of a is : ' || a);
END;
```

Output

```
a is not less than 20
value of a is : 500
PL/SQL procedure successfully completed.
```

Case statement: The **CASE statement** selects one sequence of statements to execute. However, to select the sequence, the CASE statement uses a selector rather than multiple Boolean expressions. A selector is an expression whose value is used to select one of several alternatives.

Syntax:

```
CASE [ expression ]
WHEN condition_1 THEN result_1
  WHEN condition_2 THEN result_2
  ...
  WHEN condition_n THEN result_n
 ELSE result
END
```

9. b) Write PL/SQL code to find the factorial of a given number

Ans.

```
declare
    n number;
    fac number:=1;
    i number;
begin
    n:=&n;
    for i in 1..n
    loop
    fac:=fac*i;
    end loop;
    dbms_output.put_line('factorial='||fac);
end;
/
```

Output

```
Enter value for n: 5
old 3: n:=&n;
new 3: n:=5;
factorial=120
```

10) Explain the concept of COMMIT and ROLLBACK?

Ans. COMMIT: It is used to permanently save all the transactions done by the DML commands in the database. Once issued, it cannot be undone. DBMS software implicitly uses the COMMIT command before and after every DDL command to save the change permanently in the database.

Syntax: COMMIT;

Example:

UPDATE Student SET DOB='2000-12-17' WHERE FirstName='Joe';

COMMIT;

ROLLBACK: It is used to undo the transactions that have not already been permanently saved to the database. It restores the previously stored value, i.e., the data present before the execution of the transactions.

Syntax: ROLLBACK;

Example:

UPDATE Student SET DOB='2000-12-17' WHERE FirstName='Joe';

ROLLBACK;

April/May 2015

1) Define referential integrity constraints with example.

Ans. The Referential integrity constraints is specified between two relations or tables and used to maintain the consistency among the tuples in two relations. Example: **Table1(EMP_ID, NAME, AGE, D_NO)** where **D_NO** is a foreign key. Another table such as Table2**(D_NO, D_Location)** where **D_NO** is the primary key. While inserting a tuple in Table1, the third tuple **D_NO** value should match with **D_NO** in Table2.

2) List out different types of join operations?

Ans. 1. Inner Join

- Natural Join
- Theta Join
- Equi Join

2. Outer Join

- Left Outer Join
- Right Inner Join
- Full Outer Join

3) What is group by clause? Give example.

Ans. The GROUP BY clause groups the selected rows based on identical values in a column or expression.

Example: SUM() with GROUP BY

SELECT SUM(COST) FROM PRODUCT WHERE QTY>1 GROUP BY COMPANY;

4) Mention the kind of constraints we can specify in the create command DDL?

Ans. The type of constraints that can be specified in the Create command are as follows: PRIMARY KEY, NOT NULL, UNIQUE, FOREIGN KEY, CHECK.

5) What are the advantages of PL/SQL?

Ans.
- Portability.
- Scalability.
- Manageability.
- Support for Object-Oriented Programming.

6) What is a constraint? Give the detailed explanation of key constraint and domain constraint?

Ans. Constraints are a set of rules. It is used to maintain the quality of information.

Key Constraints:

- These are called uniqueness constraints since it ensures that every tuple in the relation should be **unique**.
- A relation can have **multiple keys** or **candidate keys**(minimal superkey), out of which we choose one of the keys as primary key.
- **Null** values are not allowed in the primary key, hence Not Null constraint is also a part of key constraint.

Domain constraint

- Domain constraints specify the **set of possible values** that may be associated with an attribute.
- Such constraints may also prohibit the use of **null** values for particular attributes.
- The data types associated with domains typically include standard numeric data types for integers, string, real, time, date, currency, etc

7) (a) Explain selection and projection operation in relational algebra with an example?

Ans. **Selection Operator:** Selection Operator (σ) is a unary operator in relational algebra that performs aselection operation. It selects those rows or tuples from the relation that satisfies the selection condition.

Syntax: σ<selection_condition>(R)

Examples: Consider the following Student relation-

ID	Name	Subject	Age
100	Ashish	Maths	19
200	Rahul	Science	20
300	Naina	Physics	20
400	Sameer	Chemistry	21

Select tuples from a relation "Student" where subject is "database"

σsubject = "Science" (Student)

ID	Name	Subject	Age
200	Rahul	Science	20

Projection: Projection Operator (π) is a unary operator in relational algebra that performs a projection operation. It displays the columns of a relation or table based on the specified attributes.

Syntax: π<attribute list>(R)

Example: Consider the following Student relation-

ID	Name	Subject	Age
100	Ashish	Maths	19
200	Rahul	Science	20
300	Naina	Physics	20
400	Sameer	Chemistry	21

Result for Query **πName, Age(Student)**

Name	Age
Ashish	19
Rahul	20
Naina	20
Sameer	21

7) a) Explain insert, delete and update statements in SQL with example?

Ans. **INSERT:** It is used to insert new rows into the table. To insert values according to the table structure

Syntax: INSERT INTO table_name VALUES (value1, value2,…. value-n);

Example: INSERT INTO Student VALUES (101, 'Athen', 'Joe', 'Bangalore','07-DEC-2002');

UPDATE: It is used to update existing column value within a table. To update the columns of a table based on a condition use the following syntax.

Syntax:

UPDATE table_name SET column_1 = value1, column_2 = value2, column_n = valuen [WHERE condition]

Here, the SET statement is used to set new values to the particular column, WHERE clause is used to select rows for which the columns are updated for the given table.

Example:

UPDATE Student SET LastName = 'Dhanu', Address= 'Udupi' WHERE StudID = 102;

DELETE: It is used to delete existing records from a table, i.e., it is used to remove one or more rows from a table. The following syntax is used to delete rows from a table based on a condition

Syntax: DELETE FROM table_name [WHERE condition];

The DELETE statement only removes the data from the table, whereas the TRUNCATE statement also frees the memory along with data removal. Hence, TRUNCATE is more efficient in removing all the data from a table.

Example: DELETE FROM Student WHERE StudID=104;

8. Consider the following relation

Emp_salary(Emp-no, Ename,DOB,Dno,Salary)

Write the SQL of the following

i. Display the number of employees working in each department

ii. Find the sum of salaries of all employees.

iii. Find sum and average salaries of employee of BCA department

iv. Find the highest salary that an employee draws

v. Find the least salary that an employee draws.

-- create a table

Ans. CREATE TABLE Emp_Salary (Emp_no int PRIMARY KEY, Ename varchar(30),DOB date,Dname varchar(30), Salary int);

-- insert some values

INSERT INTO Emp_Salary VALUES (1, 'Ryan', '12-04-2000','Account',25000);

INSERT INTO Emp_Salary VALUES (2, 'Joanna', '30-11-2001','Finance',30000);

INSERT INTO Emp_Salary VALUES (3, 'Vinod', '09-10-1999','Finance',16000);

INSERT INTO Emp_Salary VALUES (4, 'Dimple', '03-01-2001','Account',28000);

INSERT INTO Emp_Salary VALUES (5, 'Sanvi', '21-07-2001','Marketing',33000);

-- fetch some values

SELECT * FROM Emp_Salary;

Emp_no	Ename	DOB	Dname	Salary
1	Ryan	12-04-2000	Account	25000
2	Joanna	30-11-2001	Finance	30000
3	Vinod	09-10-1999	Finance	16000
4	Dimple	03-01-2001	Account	28000
5	Sanvi	21-07-2001	Marketing	33000

-- Display the number of employees working in each department

SELECT Dname, COUNT(*) AS COUNT FROM Emp_Salary GROUP BY Dname;

Dname	COUNT
Account	2
Finance	2
Marketing	1

-- Find the sum of salaries of all employees.

SELECT SUM(Salary) AS SUM_OF_SALARY FROM Emp_Salary;

SUM_OF_SALARY
132000

-- Find sum and average salaries of employee of each department

SELECT Dname, SUM(Salary) AS SUM_OF_SALARY ,avg(Salary) AS AVG_OF_SALARY FROM Emp_Salary GROUP BY Dname;

Dname	SUM_OF_SALARY	AVG_OF_SALARY
Account	53000	26500
Finance	46000	23000
Marketing	33000	33000

-- Find the highest salary that an employee draws

SELECT MAX(Salary) AS MAX_SALARY FROM Emp_Salary;

MAX_SALARY
33000

-- Find the least salary that an employee draws.

SELECT MIN(Salary) AS MIN_SALARY FROM Emp_Salary;

MIN_SALARY
16000

9. a) What is cursor? What are the cursor attributes? Explain

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.

The following table specifies the cursor attributes.

Attribute	Description
%FOUND	Its return value is TRUE if DML statements like INSERT, DELETE and UPDATE affect at least one row or more rows or a SELECT INTO statement returned one or more rows. Otherwise it returns FALSE.
%NOTFOUND	Its return value is TRUE if DML statements like INSERT, DELETE and UPDATE affect no row, or a SELECT INTO statement return no rows. Otherwise it returns FALSE. It is a just opposite of %FOUND.
%ISOPEN	It always returns FALSE for implicit cursors, because the SQL cursor is automatically closed after executing its associated SQL statements.
%ROWCOUNT	It returns the number of rows affected by DML statements like INSERT, DELETE, and UPDATE or returned by a SELECT INTO statement.

9. **b) Explain for...loop statement in PL/SQL with an example.**

Ans. Execute a sequence of statements multiple times and abbreviates the code that manages the loop variable.

Syntax:

FOR counter IN initial_value .. final_value LOOP

LOOP statements;

END LOOP;

Where **initial_value** is a Start integer value and **final_value** is an End integer value

Example:

```
DECLARE
n NUMBER;
BEGIN
n:=7;
FOR i IN 1..10
LOOP
DBMS_OUTPUT.PUT_LINE (n||'*'||i||'='||i*n);
END LOOP;
END;
```

Output

```
7*1=7
7*2=14
7*3=21
7*4=28
7*5=35
7*6=42
7*7=49
7*8=56
7*9=63
7*10=70
```

May 2016

1) What are the applications of Relational algebra in RDBMS?

Ans.
- To provide a theoretical foundation for relational databases
- Particularly query languages for such databases

2) Mention the different categories of SQL statements

Ans.
- DDL – Data Definition Language.
- DML – Data Manipulation Language.
- DCL – Data Control Language.
- TCL- Transaction Control Language

3) What is an exception? Mention major types of exceptions

Ans. An error occurs during the program execution is called Exception in PL/SQL. PL/SQL facilitates programmers to catch such conditions using exception block in the program and an appropriate action is taken against the error condition. There are two types of exceptions:

- System-defined Exceptions
- User-defined Exceptions

4. a) Explain briefly schema-based constraints in relational data model?

Ans. Constraints that are directly applied in the schemas of the data model, by specifying them in the DDL. These are called as **schema-based constraints or Explicit constraints**. The schema-based constraints are as follows:

- **Domain** constraint
- **Key** constraints
- **Entity integrity** constraints
- **Referential integrity** constraints
- Constraints on **Nulls**

4. b) Explain selection and projection operation in relational algebra with an example?

Ans. **Selection Operator:** Selection Operator (σ) is a unary operator in relational algebra that performs aselection operation. It selects those rows or tuples from the relation that satisfies the selection condition.

Syntax: σ<selection_condition>(R)

Examples: Consider the following Student relation-

ID	Name	Subject	Age
100	Ashish	Maths	19
200	Rahul	Science	20
300	Naina	Physics	20
400	Sameer	Chemistry	21

Select tuples from a relation "Student" where subject is "database"

σsubject = "Science" (Student)

ID	Name	Subject	Age
200	Rahul	Science	20

Projection: Projection Operator (π) is a unary operator in relational algebra that performs a projection operation. It displays the columns of a relation or table based on the specified attributes.

Syntax: π<attribute list>(R)

Example: Consider the following Student relation-

ID	Name	Subject	Age
100	Ashish	Maths	19
200	Rahul	Science	20
300	Naina	Physics	20
400	Sameer	Chemistry	21

Result for Query **πName, Age(Student)**

Name	Age
Ashish	19
Rahul	20
Naina	20
Sameer	21

5. a) Explain briefly DDL statements with syntax and examples.

Ans. DDL stands for Data Definition Language. Data Definition Languages allow users to create, modify, and destroy the schema of database objects. All the command of DDL is auto-committed that means it permanently save all the changes in the database. DDL commands in SQL are:

CREATE: It is used to create the database or its schema objects. *To create a new table*

Syntax:

CREATE TABLE table_name (column_1 DATATYPE,column_2 DATATYPE,column_n DATATYPE);

Example:

CREATE TABLE Student (StudID int,LastName varchar(255),FirstName varchar(255),Address varchar(255));

DROP: It is used to delete the database or its schema objects. To delete an existing table

Syntax: DROP TABLE table_name;

Example: Drop table Student;

ALTER: It is used to modify the structure of the database objects. To add new column in a table

Syntax:

ALTER TABLE table_name ADD (column_1 DATATYPE,column_2 DATATYPE,column_n DATATYPE);

Example: ALTER TABLE Student ADD DOB date;

TRUNCATE: It is used to remove the whole content of the table along with the deallocation of the space occupied by the data, without affecting the table's structure. To remove data present inside a table

Syntax: TRUNCATE TABLE table_name;

Example: TRUNCATE TABLE Student;

RENAME: It is used to change the name of an existing table or a database object. To rename a table

Syntax: RENAME old_table_name TO new_table_name;

Example: RENAME Student TO STUD;

5. b) What is JOIN operation? Explain different types of join with syntax and example.

Ans. Join operation is used to combine two or more tables based on the related attributes.

Inner Join:

- Natural Join
- Theta Join
- Equi Join

Outer Join

- Left Outer Join
- Right Inner Join
- Full Outer Join

Inner Join: Inner join is a type of join in which only those tuples are selected which full fill the required conditions. All those tuples which do not satisfy the required conditions are excluded.

Natural Join(⋈): Natural Join is a join which is performed if there is a common attribute between the relations.

Notation: $R_1 \bowtie R_2$ where R_1 and R_2 are two relations.

Example: We have two tables of Student(S_id, Name, Age, C_id) and Course(C_id, C_name).

Student

S_ID	NAME	AGE	C_ID
1	Andrew	25	11
2	Dhee	30	11
3	Vandhana	31	22

Course

C_ID	C_NAME
11	Fundamentals of C
21	C++

Student ⋈ Course

S_ID	NAME	AGE	C_ID	C_NAME
1	Andrew	25	11	Fundamentals of C
2	Dhee	30	11	Fundamentals of C

Theta Join: Theta join is a join which combines the tuples from different relations according to the given theta condition. The join condition in theta join is denoted by **theta (θ)** symbol.

Notation: $R_1 \bowtie_\theta R_2$

Example: We have two tables of Student1(S_id, Name, Std, Age) and Course1 (Class, C_name)

Student1.

S_ID	NAME	STD	AGE
1	Andrew	10	25
2	Dhee	9	30
3	Vandhana	8	31

Course1

CLASS	C_NAME
8	Fundamentals of C
9	C++

Now, we will perform theta join on both the tables

Student1 $\bowtie_{\text{Student1.std=Course1.class}}$ **Course1**

S_ID	NAME	STD	AGE	CLASS	C_NAME
2	Dhee	9	30	9	C++
3	Vandhana	8	31	8	Fundamentals of C

Equi Join: Equi Join is a type of theta join where we use only the equality operator.

Outer Join: In outer join, we include those tuples which meet the given condition along with that, we also add those tuples which do not meet the required condition. The result also includes the tuples from the left and right tables which do not satisfy the conditions.

Left Outer Join(⟕): Left Outer Join is a type of join in which all the tuples from left relation are included and only those tuples from right relation are included which have a common value in the common attribute on which the join is being performed.

Notation: $R_1 ⟕ R_2$ where R_1 and R_2 are relations.

Example: We use the above two tables Student(S_id, Name, Age, C_id) and Course(C_id, C_name). Now, we will perform left outer join on both the tables i.e

Student ⋈ Course

S_ID	NAME	AGE	C_ID	C_NAME
1	Andrew	25	11	Fundamentals of C
2	Dhee	30	11	Fundamentals of C
3	Vandhana	31	22	---

Right Outer Join(⟖): Right Outer Join is a type of join in which all the tuples from right relation are included and only those tuples from left relation are included which have a common value in the common attribute on which the right join is being performed.

Notation: $R_1 ⟖ R_2$ where R_1 and R_2 are relations.

Example: We use the above two tables Student(S_id, Name, Age, C_id) and Course(C_id, C_name). Now, we will perform right outer join on both the tables i.e

Student ⋈ Course

S_ID	NAME	AGE	C_ID	C_NAME
1	Andrew	25	11	Fundamentals of C
2	Dhee	30	11	Fundamentals of C
---	----	----	21	C++

Full Outer Join(⟗): Full Outer Join is a type of join in which all the tuples from the left and right relation which are having the same value on the common attribute.

Notation: $R_1 ⟗ R_2$ where R_1 and R_2 are relations.

Example: We use the above two tables Student(S_id, Name, Age, C_id) and Course(C_id, C_name). Now, we will perform full outer join on both the tables i.e

Student ⋈ Course

S_ID	NAME	AGE	C_ID	C_NAME
1	Andrew	25	11	Fundamentals of C
2	Dhee	30	11	Fundamentals of C
3	Vandhana	31	22	---
----	----	----	21	C++

6. a) What is a database trigger? Explain any four types of triggers?

Ans. Triggers are stored programs, which are automatically executed or fired when some event occurs. Triggers are 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).
- A database operation (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN).

Triggers could be defined on the table, view, schema, or database with which the event is associated.

Syntax

```
CREATE [OR REPLACE ] TRIGGER trigger_name
{BEFORE | AFTER | INSTEAD OF }
{INSERT [OR] | UPDATE [OR] | DELETE}
[OF col_name]
ON table_name
[REFERENCING OLD AS o NEW AS n]
[FOR EACH ROW]
WHEN (condition)
DECLARE
  Declaration-statements
BEGIN
  Executable-statements
EXCEPTION
  Exception-handling-statements
END;
```

- CREATE [OR REPLACE] TRIGGER trigger_name: It creates or replaces an existing trigger with the trigger_name.
- {BEFORE | AFTER | INSTEAD OF} : This specifies when the trigger would be executed. The INSTEAD OF clause is used for creating trigger on a view.
- {INSERT [OR] | UPDATE [OR] | DELETE}: This specifies the DML operation.
- [OF col_name]: This specifies the column name that would be updated.
- [ON table_name]: This specifies the name of the table associated with the trigger.

- [REFERENCING OLD AS o NEW AS n]: This allows you to refer new and old values for various DML statements, like INSERT, UPDATE, and DELETE.
- [FOR EACH ROW]: This specifies a row level trigger, i.e., the trigger would be executed for each row being affected. Otherwise the trigger will execute just once when the SQL statement is executed, which is called a table level trigger.

6. b) Explain While... Loop statement in PL/SQL with an example?

Ans. Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.

WHILE <condition>

LOOP statements;

END LOOP;

Example:

```
DECLARE
i INTEGER := 1;
sum INTEGER := 0;
BEGIN
WHILE i <= 10 LOOP
DBMS_OUTPUT.PUT_LINE(i);
sum:=sum+I;
i := i+1;
END LOOP;
dbms_output.put_line('Sum is ' || sum);
END;
```

Output:

```
1
2
3
4
5
Sum is 15
```

May 2017

1. Explain Domain and Tuple?

Ans. A domain is referred to in a relation schema by the **domain name** and has a set of associated values. Each row in the relation is known as tuple.

2. Explain Commit and Rollback commands

Ans. **COMMIT:** It is used to permanently save all the transactions done by the DML commands in the database.

ROLLBACK: It is used to undo the transactions that have not already been permanently saved to the database.

3. What is database Triggers?

Ans. Triggers are stored programs, which are automatically executed or fired when some event occurs.

4. a) Explain different characteristics of relations.

Ans. **Ordering of Tuples in a Relation**: A relation is defined as a set of tuples. The definition of a relation does not specify any order. There is no preference for one ordering over another. However, in a file, records are physically stored on disk, so there always is an order among the records. This ordering indicates first, second, i^{th}, and last records in the file.

Ordering of attributes in a relation R:

The ordering of attributes is not important, because the attribute name appears with its value. Attributes in $R(A_1, A_2, ..., An)$ and the values in $t=<v_1, v_2, ..., v_n>$ to be ordered .

Values in a tuple: All values are considered atomic. A special **null** value is used to represent values that are unknown or inapplicable to certain tuples.

4. b) Explain cartesian product and selection operations.

Ans. **Cartesian Product (X):** Cross product between two relations let say A and B, so cross product between A X B will results all the attributes of A followed by each attribute of B. Each record of A will pairs with every record of B.

Table A

Name	Age	Sex
Ram	24	Male
Abhi	22	Female
Keerthana	23	Female

Table B

ID	Course
1	JAVA
2	DS

A × B

Name	Age	Sex	ID	Course
Ram	24	Male	1	JAVA
Ram	24	Male	2	DS
Abhi	22	Female	1	JAVA
Abhi	22	Female	2	DS
Keerthana	23	Female	1	JAVA
Keerthana	23	Female	2	DS

Selection Operator: Selection Operator (σ) is a unary operator in relational algebra that performs aselection operation. It selects those rows or tuples from the relation that satisfies the selection condition.

Syntax: σ<selection_condition>(R)

Examples: Consider the following Student relation-

ID	Name	Subject	Age
100	Ashish	Maths	19
200	Rahul	Science	20
300	Naina	Physics	20
400	Sameer	Chemistry	21

Select tuples from a relation "Student" where subject is "database"

$\sigma_{subject}$ = "Science" (Student)

ID	Name	Subject	Age
200	Rahul	Science	20

Select tuples from a relation "Books" where subject is "Physics" and Age is "20"

$\sigma_{subject}$ = "Physics" Age = "20" (Student)

5. a) Write an SQL query for the following:

i. To create a table of Hospital database with minimum 5 fields

ii. To insert two records

iii. To add new field

iv. To display all records

Ans. --Create table

CREATE TABLE PATIENT (PID int PRIMARY KEY, Name varchar(30),Gender varchar(10),Address varchar(10),Phone numeric(10));

--insert 2 records;

INSERT INTO PATIENT(PID,Name,Gender, Address, Phone)VALUES(1901,'Anita','Female', 'Hebbal', 9933890756);

INSERT INTO PATIENT(PID,Name,Gender, Address,Phone)VALUES(1902,'Leena','Female', 'Bangalore', 9023985673);

select * from PATIENT;

PID	Name	Gender	Address	Phone
1901	Anita	Female	Hebbal	9933890756
1902	Leena	Female	Bangalore	9023985673

--Add new field

ALTER TABLE PATIENT ADD DOB date;

--Display the table

select * from PATIENT;

PID	Name	Gender	Address	Phone	DOB
1901	Anita	Female	Hebbal	9933890756	NULL
1902	Leena	Female	Bangalore	9023985673	NULL

5. b) Explain different types of cursors?

Ans. 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**.

One 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.

The cursor attributes are %FOUND, %NOTFOUND, %ISOPEN, %ROWCOUNT

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.

The syntax for creating an explicit cursor is

CURSOR cursor_name **IS** select_statement;

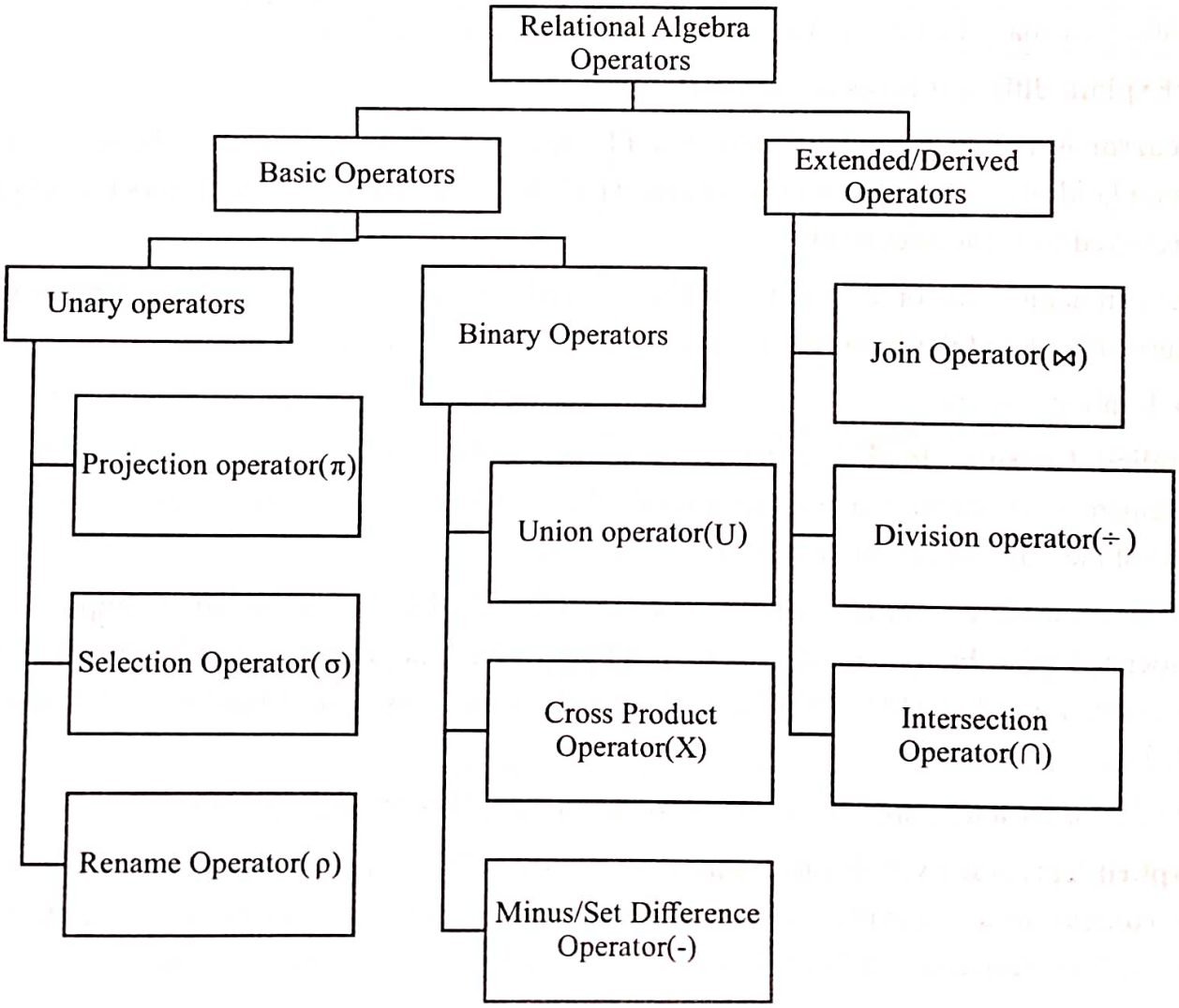
Working with an explicit cursor includes the following steps –

- 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

May 2018

1. Explain Relation algebra in detail?

Ans. Relational algebra is a procedural query language, which takes relation as input and generate relation as output. The basic set of operations for the relational model is the relational algebra. Relational Operators always work on one or more relational tables. The operators in relational algebra are classified as

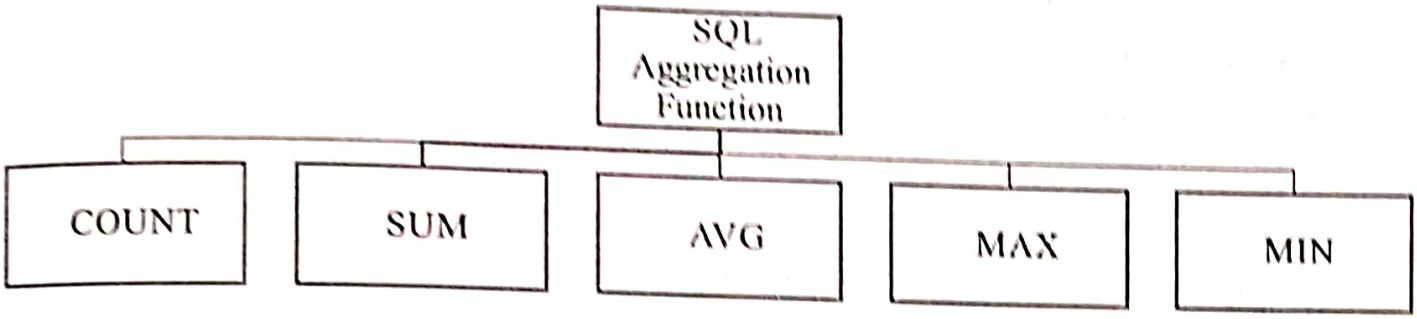


Brief all operators in detail

2. a) Explain different aggregate functions in SQL with syntax and examples

Ans. SQL aggregation function is used to perform the calculations on multiple rows of a single column of a table. It returns a single value.

Types of SQL Aggregation Function



COUNT Function: COUNT function is used to Count the number of rows in a database table.

Syntax: COUNT(*) or COUNT([ALL|DISTINCT] expression)

Example: Consider the PRODUCT table

PRODUCT	COMPANY	QTY	RATE	COST
Keyboard	ABC	2	700	1400
Mouse	ABC	4	600	2400
Processor	Intel	2	5600	11200
Printer	HP	2	3000	6000

Example: COUNT()

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

Output:

10

Example: COUNT with WHERE

```
SELECT COUNT(*) FROM PRODUCT WHERE RATE>=1000;
```

Output:

2

SUM Function: Sum function is used to calculate the sum of all selected columns. It works on numeric fields only.

Syntax: SUM() or SUM([ALL|DISTINCT] expression)

Example: SUM()

```
SELECT SUM(COST) FROM PRODUCT;
```

Output:

21000

AVG function: The AVG function is used to calculate the average value of the numeric type. AVG function returns the average of all non-Null values.

Syntax: AVG() or AVG([ALL|DISTINCT] expression)

Example: SELECT AVG(COST) FROM PRODUCT;

Output: 5250

MAX Function: MAX function is used to find the maximum value of a certain column. This function determines the largest value of all selected values of a column.

Syntax: MAX() or MAX([ALL|DISTINCT] expression)

Example: SELECT MAX(RATE) FROM PRODUCT;

Output: 5600

MIN Function: MIN function is used to find the minimum value of a certain column. This function determines the smallest value of all selected values of a column.

Syntax: MIN() or MIN([ALL|DISTINCT] expression)

Example: SELECT MIN(RATE) FROM PRODUCT;

Output: 600

2. b) What are joins? Explain INNER JOIN and OUTER JOIN

Ans. Join operation is used to combine two or more tables based on the related attributes.

Inner Join

- Natural Join
- Theta Join
- Equi Join

Outer Join

- Left Outer Join
- Right Inner Join
- Full Outer Join

Inner Join: Inner join is a type of join in which only those tuples are selected which full fill the required conditions. All those tuples which do not satisfy the required conditions are excluded.

Natural Join(⋈): Natural Join is a join which is performed if there is a common attribute between the relations.

Notation: $R_1 \bowtie R_2$ where R_1 and R_2 are two relations.

Example: We have two tables of Student(S_id, Name, Age, C_id) and Course(C_id, C_name).

Student

S_ID	NAME	AGE	C_ID
1	Andrew	25	11
2	Dhee	30	11
3	Vandhana	31	22

Course

C_ID	C_NAME
11	Fundamentals of C
21	C++

Student ⋈ Course

S_ID	NAME	AGE	C_ID	C_NAME
1	Andrew	25	11	Fundamentals of C
2	Dhee	30	11	Fundamentals of C

Theta Join: Theta join is a join which combines the tuples from different relations according to the given theta condition. The join condition in theta join is denoted by **theta (θ)** symbol.

Notation: $R_1 \bowtie_\theta R_2$

Example: We have two tables of Student1(S_id, Name, Std, Age) and Course1 (Class, C_name)

Student1.

S_ID	NAME	STD	AGE
1	Andrew	10	25
2	Dhee	9	30
3	Vandhana	8	31

Course1

CLASS	C_NAME

Now, we will perform theta join on both the tables

Student1 $\bowtie_{(Student1.std=Course1.class)}$ **Course1**

S_ID	NAME	STD	AGE	CLASS	C_NAME
2	Dhee	9	30	9	C++
3	Vandhana	8	31	8	Fundamentals of C

Equi Join: Equi Join is a type of theta join where we use only the equality operator.

Outer Join: In outer join, we include those tuples which meet the given condition along with that, we also add those tuples which do not meet the required condition. The result also includes the tuples from the left and right tables which do not satisfy the conditions.

Left Outer Join(⟕): Left Outer Join is a type of join in which all the tuples from left relation are included and only those tuples from right relation are included which have a common value in the common attribute on which the join is being performed.

Notation: $R_1 \leftouterjoin R_2$ where R_1 and R_2 are relations.

Example: We use the above two tables Student(S_id, Name, Age, C_id) and Course(C_id, C_name). Now, we will perform left outer join on both the tables i.e

Student1 ⋈ Course

S_ID	NAME	AGE	C_ID	C_NAME
1	Andrew	25	11	Fundamentals of C
2	Dhee	30	11	Fundamentals of C
3	Vandhana	31	22	---

Right Outer Join(⟖): Right Outer Join is a type of join in which all the tuples from right relation are included and only those tuples from left relation are included which have a common value in the common attribute on which the right join is being performed.

Notation: $R_1 ⟖ R_2$ where R_1 and R_2 are relations.

Example: We use the above two tables Student(S_id, Name, Age, C_id) and Course(C_id, C_name). Now, we will perform right outer join on both the tables i.e

Student ⋈ Course

S_ID	NAME	AGE	C_ID	C_NAME
1	Andrew	25	11	Fundamentals of C
2	Dhee	30	11	Fundamentals of C
—	----	----	21	C++

Full Outer Join(⟗): Full Outer Join is a type of join in which all the tuples from the left and right relation which are having the same value on the common attribute.

Notation: $R_1 ⟗ R_2$ where R_1 and R_2 are relations.

Example: We use the above two tables Student(S_id, Name, Age, C_id) and Course(C_id, C_name). Now, we will perform full outer join on both the tables i.e

Student ⋈ Course

S_ID	NAME	AGE	C_ID	C_NAME
1	Andrew	25	11	Fundamentals of C
2	Dhee	30	11	Fundamentals of C
3	Vandhana	31	22	---
----	----	----	21	C++

3. a) Explain briefly DDL statements with syntax and examples

Ans. DDL stands for Data Definition Language. Data Definition Languages allow users to create, modify, and destroy the schema of database objects. All the command of DDL is auto-committed that means it permanently save all the changes in the database.

DDL commands in SQL are:

CREATE: It is used to create the database or its schema objects. *To create a new*

Syntax:

CREATE TABLE table_name (column_1 DATATYPE,column_2 DATATYPE,column_n DATATYPE);

Example:

CREATE TABLE Student (StudID int,LastName varchar(255),FirstName varchar(255),Address varchar(255));

DROP: It is used to delete the database or its schema objects. To delete an existing table

Syntax: DROP TABLE table_name;

Example: Drop table Student;

ALTER: It is used to modify the structure of the database objects. To add new column in a table

Syntax:

ALTER TABLE table_name ADD (column_1 DATATYPE,column_2 DATATYPE,column_n DATATYPE);

Example: ALTER TABLE Student ADD DOB date;

TRUNCATE: It is used to remove the whole content of the table along with the deallocation of the space occupied by the data, without affecting the table's structure. To remove data present inside a table

Syntax: TRUNCATE TABLE table_name;

Example: TRUNCATE TABLE Student;

RENAME: It is used to change the name of an existing table or a database object. To rename a table

Syntax: RENAME old_table_name TO new_table_name;

Example: RENAME Student TO STUD;

3. **b) Create an Employee database using the following fields**

FIELD NAME	DATA TYPE
EMPNO	NUMBER
ENAME	CHAR
DOB	DATE
DEPT	STRING
SALARY	REAL

i. create the table

ii. Enter 5 tuples

iii. Find sum of salaries of all employees

iv. Find highest and least salaries of all employees

– create a table

Ans. CREATE TABLE EMPLOYEE(EMPNO NUMBER PRIMARY KEY, ENAME CHAR,DOB date,DEPT STRING,Salary REAL);

-- insert some values

INSERT INTO EMPLOYEE VALUES (1, 'Ryan', '12-04-2000','Account',25000);

INSERT INTO EMPLOYEE VALUES (2, 'Joanna', '30-11-2001','Finance',30000);

INSERT INTO EMPLOYEE VALUES (3, 'Vinod', '09-10-1999','Finance',16000);

INSERT INTO EMPLOYEE VALUES (4, 'Dimple', '03-01-2001','Account',28000);

INSERT INTO EMPLOYEE VALUES (5, 'Sanvi', '21-07-2001','Marketing',33000);

-- fetch some values

SELECT * FROM EMPLOYEE;

EMP NO	ENAME	DOB	DEPT	SALARY
1	Ryan	12-04-2000	Account	25000.0
2	Joanna	30-11-2001	Finance	30000.0
3	Vinod	09-10-1999	Finance	16000.0
4	Dimple	03-01-2001	Account	28000.0
5	Sanvi	21-07-2001	Marketing	33000.0

-- Find the sum of salaries of all employees.

SELECT SUM(Salary) AS SUM_OF_SALARY FROM EMPLOYEE;

SUM_OF_SALARY
132000

-- Find the highest salary that an employee draws

SELECT MAX(Salary) AS MAX_SALARY,MIN(Salary) AS MIN_SALARY FROM EMPLOYEE;

MAX_SALARY	MIN_SALARY
33000	16000

4. Explain different data types in SQL.

Ans. Datatypes are mainly classified into

- String Data types
- Numeric Data types
- Date and time Data types

5. Expand PL/SQL. Mention any two advantages.

Ans. PL/SQL stands for procedural language extension to Structured Query Language. The advantages are:

- Portability.
- Scalability.

6. What is a view? Give the syntax for view creation.

Ans. Views in SQL are considered as a virtual table. A view also contains rows and columns. To create the view, we can select the fields from one or more tables present in the database.

Example:

```
CREATE VIEW DetailsView AS SELECT NAME, ADDRESS FROM Student_Details
WHERE STU_ID < 4;
```