Term work No.1

Entity-Relationship Model

Objective of the experiment: To ensure that you get a precise understanding of the nature of the data and how the enterprise uses it, you need to have a universal model for interaction that is non-technical and free of ambiguities and easily readable to both technical as well as non-technical members.

Theory:

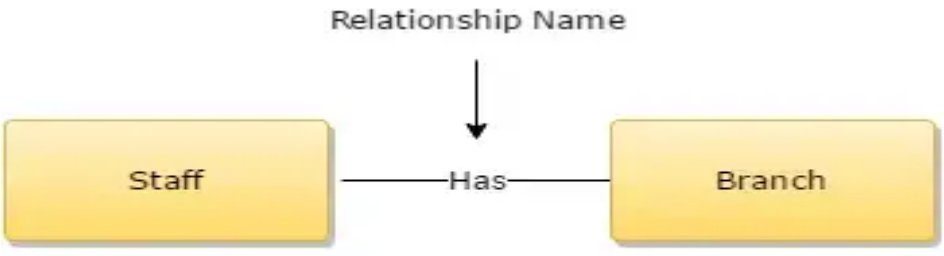
What is ER diagram?

ER-Diagram is a pictorial representation of data that describes how data is communicated and related to each other.

- **Entities:** They are represented using the rectangle-shaped box. These rectangles are named with the entity set they represent**.**

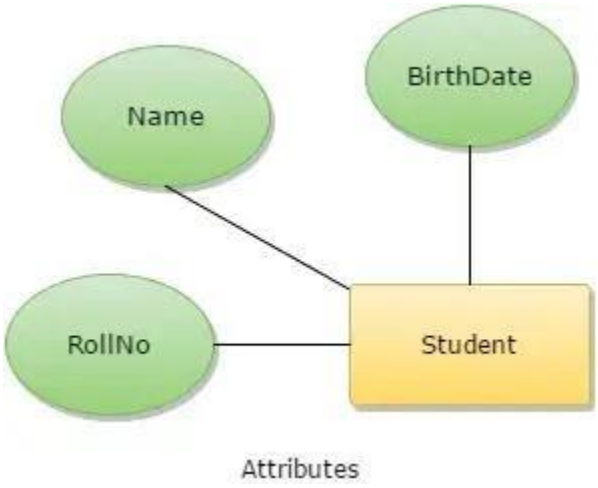
Entities

- **Entity type:** It is a group of objects with the same properties that are identified by the enterprise as having an independent existence.
- **Relationship type:** A relationship type is a set of associations between one or more participating entity types. Each relationship type is given a name that describes itsfunction.



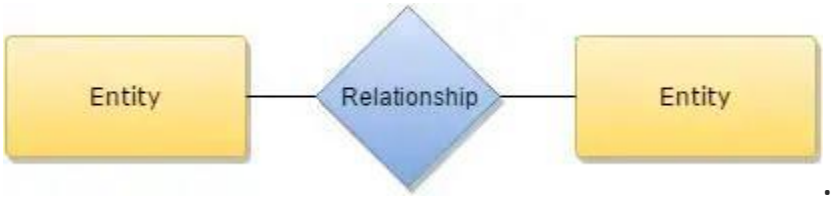
Branch has staff

- **Degree of the relationship:** The entities occupied in a particular relationship type are referred to as participants in that relationship. The number of participants involved in a relationship type is termed as the degree of that relationship.
- **Attributes:** The properties of entities that are represented using ellipse-shaped figures. Every elliptical figure represents one attribute and is directly connected to its entity (which is represented as a rectangle).



Attributes

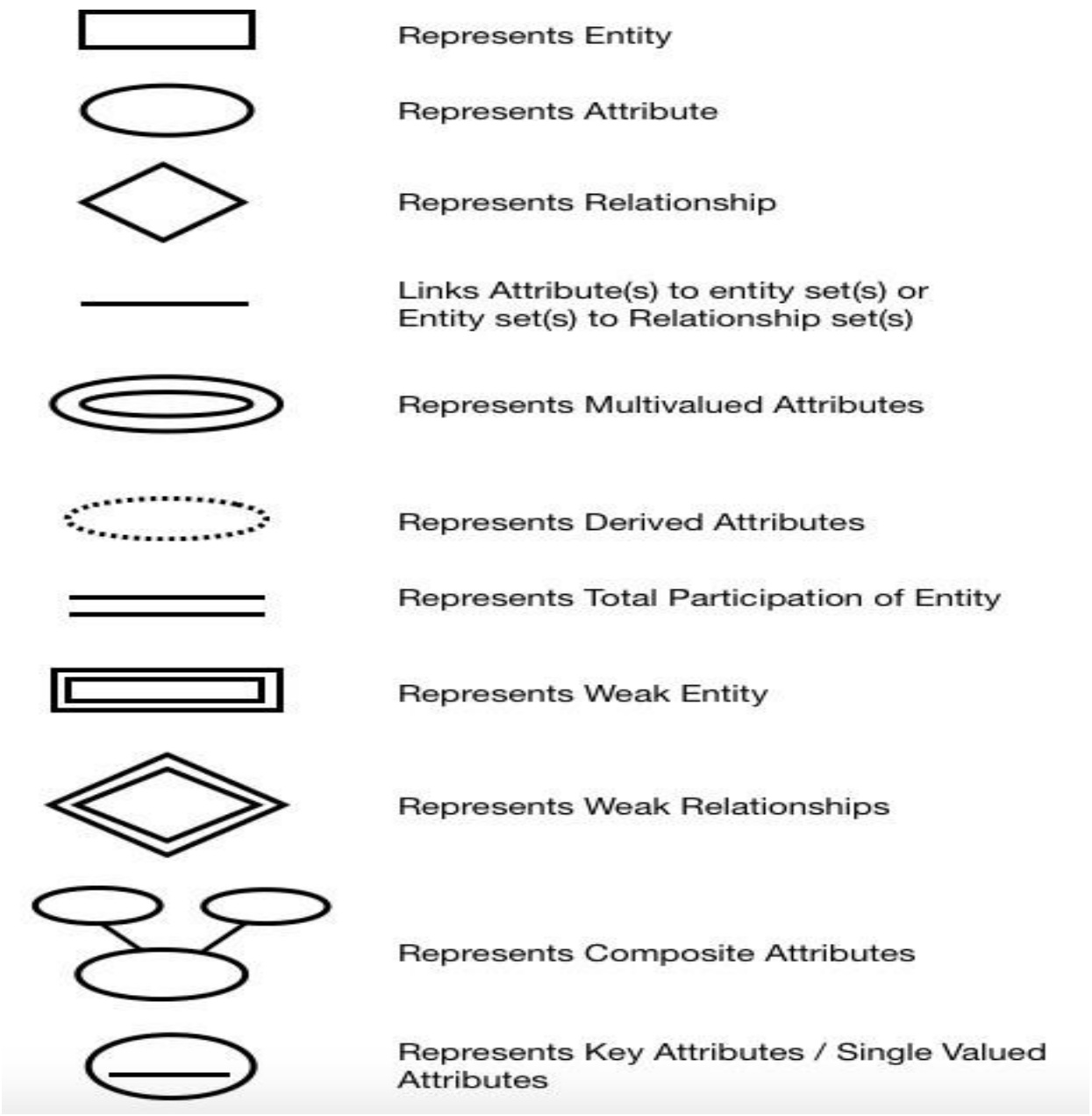
- **Relationship:** A diamond-shaped box represents relationships. All the entities (rectangle-shaped) participating in a relationship get connected using a line



There are four types of relationships. These are:

1. One-to-one: When only a single instance of an entity is associated with the relationship, it is termed as '1:1'.
2. One-to-many: When more than one instance of an entity is related and linked with a relationship, it is termed as '1:N'.
3. Many-to-one: When more than one instance of an entity is linked with the relationship, it is termed as 'N:1'.
4. Many-to-many: When more than one instance of an entity on the left and more than one instance of an entity on the right can be linked with the relationship, thenit is termed as N:N relationship

Notations


Represents Entity
Represents Attribute
Represents Relationship
Links Attribute(s) to entity set(s) or
Entity set(s) to Relationship set(s)
Represents Multivalued Attributes
Represents Derived Attributes
Represents Total Participation of Entity
Represents Weak Entity
Represents Weak Relationships
Represents Composite Attributes
Represents Key Attributes / Single Valued
Attributes

Application of the experiment performed: (TO BE WRITTEN BY STUDENT)

[***PLEASE ASK STUDENTS TO WRITE THE GIVEN PROBLEM DEFINITION, IDENTIFY ENTITIES, ATTRIBUTES, RELATIONS ETC***]

Procedure/Algorithm

[STUDENTS ARE SUPPOSED TO WRITE THE STEPS INVOLVED IN DESIGNING THEER DIAGRAM]

Output and Observation:

[PRINTOUT OF ER DIAGRAM TO BE ATTACHED]

Outcome of the experiment: At the end of the term work, the student will be able to design an ER-diagram for the given application.

Term work No.2

ER-Relational Mapping Rules

Objective of the experiment:

To convert ER diagram to Relational Model which can directly be implemented by any RDBMS like Oracle, MySQL etc.

Theory:

ER models are converted into the relational model because RDBMS can easily implement relational models like Oracle, My SQL, etc.

ER diagrams mainly consist of

- Entity: An entity is a real-world object having some attributes.
- Relationship: It is an association among different entities.

  Set of different rules are applied to convert the ER diagram to relational schema.\

  1. Mapping of regular entities
  2. Mapping of weak entities.
  3. Mapping of binary 1:1 relationship type.
  4. Mapping of binary 1: N relationship type.
  5. Mapping of binary M: N relationship types.
  6. Mapping of multivalued attributes.
  7. Mapping of N-ary relationship types.

Application of the experiment performed:

[***PLEASE ASK STUDENTS TO WRITE THE GIVEN PROBLEM DEFINITION, IDENTIFY ENTITIES, ATTRIBUTES, RELATIONS ETC***]

Procedure/Algorithm

[WRITE THE ALGORITHM TO CONVERT ER TO RELATIONAL SCHEMA AND APPLYEVERY STEP TO THE GIVEN PROBLEM DEFINITION]

Output and Observation:

[*ATTACH THE PRINTOUT OF SCHEMA DIAGRAM FOR THE GIVEN PROBLEM DEFINITION]*

Outcome of the experiment: At the end of the term work, the student will be able to convert an ER-diagram into relational schema for the given application.

Term work No.3

Relational Algebra Operations

Objective of the Experiment: Its primary objective is to provide a foundation for understanding and developing database queries and operations

Relational Algebra is a procedural query language that takes relations as an input and returns relations as an output. Relational algebra mainly provides a theoretical foundation for relational databases and SQL.

Given that these operators accept relations as input and produce relations as output, they can be combined and used to express potentially complex queries that transform potentially many input relations *(whose data are stored in the database)* into a single output relation *(the query results).*

Relational Algebra is very important for several reasons:

1. Provides a foundation for relational model operations.

2. It is used as a basis for implementing and optimizing queries in relational database management systems (RDBMSs).

3. Some of its concepts are incorporated into the SQL for RDBMSs.

Relational Algebra consists of several groups of operations

1. Unary Relational Operations

(i)SELECT (symbol: **σ** (sigma)) (ii)PROJECT (symbol: **π**(pi))

2. Relational Algebra Operations from Set Theory

(i)UNION (∪) (ii) INTERSECTION (∩) (iii)DIFFERENCE (or MINUS, –) (iv)CARTESIAN PRODUCT (×)

3. Binary Relational Operations
JOIN (several variations of JOIN exist)

4. Additional Relational Operations

(i) OUTER JOINS

(ii) AGGREGATE FUNCTIONS (These compute a summary of information: for example, SUM, COUNT, AVG, MIN, MAX).

SELECT (σ (sigma))

- Used to select a *subset* of the tuples from a relation based on a **selection condition.**
- The selection condition acts as a **filter.** Keeps only those tuples that satisfy the qualifying condition
- Tuples satisfying the condition are *selected* whereas the other tuples are discarded (*filtered out*).
- Can be viewed as horizontal partitioning (one of tuples
  In general, the *select* operation is denoted by

$$\sigma_{\text{<selection condition>}}(R)$$

where

- the symbol **σ** (sigma) is used to denote the *select* operator,
- the selection condition is a Boolean (conditional) expression specified on the attributes of relation R
- tuples that make the condition **true** are selected ***(appear in the result of the operation)***
- tuples that make the condition **false** are filtered out ***(discarded from the result of the operation)***
- <selection condition> is made up of a number of clauses of the form <attribute name> <comparison op> <constant value> ***OR*** <attribute name> <comparison op> < attribute name >

SELECT Operation Properties

- The SELECT operation produces a relation S that has the same schema (same attributes)

as R

- SELECT operation is commutative. i.e

  $\sigma_{<condition1>}(\sigma_{<condition2>}(R)) = \sigma_{<condition2>}(\sigma_{<condition1>}(R))$

- The number of tuples in the result of a SELECT is less than (or equal to) the number of tuples in the input relation R.

PROJECT (π (Pi))

- This operation keeps certain *columns* (attributes) from a relation and discards the other columns.
- PROJECT creates vertical partitioning.
- The list of specified columns (attributes) is kept in each tuple. The other attributes in each tuple are discarded.

  In general, the *project* operation is denoted by

$$\pi_{<attribute\ list>}(R)$$

  where

  π used to represent the PROJECT operation.

  <attribute list> is the desired list of attributes from the attributes of relation R.
- The result of the PROJECT operation has only the attributes specified in <attribute list> in the same order as they appear in the list.
- The project operation *removes any duplicate tuples.* This is because the result of the *project* operation must be a *set of tuples.* Mathematical sets *do not allow* duplicateelements.
  - **PROJECT Operation Properties:**
    - ✓ The number of tuples in the result of projection $\pi_{<list>}(R)$ is always less or equal to the number of tuples in R.
    - ✓ If the list of attributes includes a *key* of R, then the number of tuples in the result of PROJECT is *equal* to the number of tuples in R
    - ✓ PROJECT is *not* commutative.

RENAME (ρ(rho))

- The general RENAME operation ρ can be expressed by any of the following forms:

  $\rho_{S(B1, B2, \ldots, Bn)}(R)$ changes both:

  the relation name to S, *and* the column (attribute) names to B1, B1, …..Bn

  $\rho_S(R)$ changes:

  the *relation's name* only to S

  $\rho_{(B1, B2, \ldots, Bn)}(R)$ changes:

  the *column (attribute) names* only to B1, B1, …. Bn

Relational Algebra Operations from Set Theory

1. UNION
2. INTERSECTION
3. SET DIFFERENCE
4. CARTESIAN PRODUCT

UNION

- Union operation is a Binary operation, denoted by ∪
- The result of R ∪ S, is a relation that includes all tuples that are either in R or in S or both R and S
- Duplicate tuples are eliminated
- The two-operand relations R and S must be "type compatible" (or UNION compatible)
- R and S must have the same number of attributes
- Each pair of corresponding attributes must be type compatible (have the same or compatible domains)

INTERSECTION

- INTERSECTION is denoted by ∩.
- The result of the operation R ∩ S, is a relation that includes all tuples that are in both R and S.
- The attribute names in the result will be the same as the attribute names in R.
- The two-operand relations R and S must be "type compatible".

SET DIFFERENCE

- SET DIFFERENCE (also called MINUS or EXCEPT) is denoted by –
- The result of R – S, is a relation that includes all tuples that are in R but not in S.
- The attribute names in the result will be the same as the attribute names in R
- The two-operand relations R and S must be "type compatible".

Type compatibility: Two relations A(P1, P2, …, Pn) and B(Q1, Q2, …, Qn) are said to be Type compatible (or Union compatible) if both the relation have the same degree 'k' and domain(Pi) = domain(Qi) for 1<= i <= k.

CARTESIAN PRODUCT

- CARTESIAN (or CROSS) PRODUCT Operation
- This operation is used to combine tuples from two relations in a combinatorial fashion.
- Denoted by R(A1, A2, . . ., An) x S(B1, B2, . . ., Bm)
- Result is a relation Q with degree n + m attributes:
- Q(A1, A2, . . ., An, B1, B2, . . ., Bm), in that order.
- The resulting relation state has one tuple for each combination of tuples—one from R and one from S.
- Hence, if R has nR tuples (denoted as |R| = nR), and S has nS tuples, then R x S will have nR * nS tuples.
- The two operands do NOT have to be "type compatible".

JOIN

- JOIN Operation (denoted by ⋈)
- it allows us to *combine related tuples* from various relations
- The general form of a join operation on two relations R(A1, A2, . . ., An) and S(B1, B2, . . ., Bm) is:

$$R_{<\text{join-condition}>}S$$

where R and S can be any relations that result from general *relational algebraexpressions*.

Some properties of JOIN

- Consider the following JOIN operation:
- R(A1, A2, . . ., An) ⋈ S(B1, B2, . . ., Bm)

R. Ai =S. Bj

- The result is a relation Q with degree n + m attributes:
- Q (A1, A2, . . ., An, B1, B2, . . ., Bm), in that order.
- The resulting relation state has one tuple for each combination of tuples—r from R and s from S, but *only if they satisfy the join condition* r[Ai]=s[Bj]
- Hence, if R has n_R tuples, and S has n_S tuples, then the join result will generally have *less than* $n_R * n_S$ tuples.
- Only related tuples (based on the join condition) will appear in the result.
- The different types of JOINS are: **Equi Join , Natural Join** and **Outer Joins(Left outer Join, Right Outer Join, Full Outer Join).**

Application of the experiment performed: (TO BE WRITTEN BY STUDENT)

[***PLEASE ASK STUDENTS TO WRITE THE GIVEN PROBLEM DEFINITION]***

Procedure/Algorithm

[STUDENTS ARE SUPPOSED TO WRITE THE STEPS TO CREATE TABLE, INSERTVALUES INTO TABLE AND WRITE QUERIES]

Output and Observation:

[THE SOLUTIONS TO BE WRITTEN BY STUDENTS FOR THE QUERIES GIVEN]

Outcome of the experiment: At the end of the term work, the student will be able to explain the relational operators and write the expressions.

Term work No.4

Normalization

Objective of the experiment: The main objective of database normalization is to eliminate redundant data, minimize data modification errors, and simplify the query process. Ultimately, normalization goes beyond simply standardizing data, and can even improve workflow, increase security, and lessen costs.

Theory:

Normalization

A large database defined as a single relation may result in data duplication. This repetition of data may result in:

- Making relations very large.
- It isn't easy to maintain and update data as it would involve searching many records in relation.
- Wastage and poor utilization of disk space and resources.
- The likelihood of errors and inconsistencies increases.

What is Normalization?

- Normalization is the process of organizing the data in the database.
- Normalization is used to minimize the redundancy from a relation or set of relations. It is also used to eliminate undesirable characteristics like Insertion, Update, and Deletion Anomalies.
- Normalization divides the larger table into smaller and links them using relationships.
- The normal form is used to reduce redundancy from the database table.

Why do we need Normalization?

The main reason for normalizing the relations is removing these anomalies. Failure to eliminate anomalies leads to data redundancy and can cause data integrity and other problems as the database grows. Normalization consists of a series of guidelines that helps to guide you in creating a good database structure.

Data modification anomalies can be categorized into three types:

Insertion Anomaly: Insertion Anomaly refers to when one cannot insert a new tuple intoa relationship due to lack of data.

- **Deletion Anomaly:** The delete anomaly refers to the situation where the deletion of data results in the unintended loss of some other important data.
- **Updatation Anomaly:** The update anomaly is when an update of a single data value requires multiple rows of data to be updated.

Types of Normal Form

Normal Form	Description
1NF	A relation is in 1NF if it contains an atomic value.
2NF	A relation will be in 2NF if it is in 1NF and all non-key attributes are fully functional dependent on the primary key.
3NF	A relation will be in 3NF if it is in 2NF and no transition dependency exists.
BCNF	A stronger definition of 3NF is known as Boyce Codd's normal form

First Normal Form (1NF)

- A relation will be 1NF if it contains an atomic value.
- It states that an attribute of a table cannot hold multiple values. It must hold only single-valued attribute.
- First normal form disallows the multi-valued attribute, composite attribute, and their combinations.

Second Normal Form (2NF)

- In the 2NF, relational must be in 1NF.
- In the second normal form, all non-key attributes are fully functional dependent on the primary key

Third Normal Form (3NF)

- A relation will be in 3NF if it is in 2NF and not contain any transitive partial dependency.
- 3NF is used to reduce the data duplication. It is also used to achieve the data integrity.
- If there is no transitive dependency for non-prime attributes, then the relation must be in third normal form.

A relation is in third normal form if it holds atleast one of the following conditions for every non-trivial function dependency $X \rightarrow Y$.

1. X is a super key.
2. Y is a prime attribute, i.e., each element of Y is part of some candidate key.

Advantages of Normalization

- Normalization helps to minimize data redundancy.
- Greater overall database organization.
- Data consistency within the database.
- Much more flexible database design.
- Enforces the concept of relational integrity.

Application of the experiment performed: (TO BE WRITTEN BY STUDENT)

[***PLEASE ASK STUDENTS TO WRITE THE GIVEN PROBLEM DEFINITION]***

Procedure/Algorithm

[STUDENTS ARE SUPPOSED TO WRITE THE STEPS TO CREATE TABLE, INSERTVALUES INTO TABLE AND WRITE QUERIES]

Output and Observation:

[PRINTOUT OF RESULTS OF CREATE STATEMENT, INSERT STATEMENTS ANDQUERIES TO BE ATTACHED]

Outcome of the Experiment: The student will be able to analyse the given database applications using E-R diagrams and apply the normalization to produce schema diagrams and relations.

Term work No.5

DDL and DML Commands

Objective of the experiment: SQL experiment is to equip students with the necessary skills and knowledge to effectively interact with and manipulate relational databases using SQL queries. This practical experience helps students gain a deeper understanding of the theoretical concepts covered in the DBMS course and prepares them for real-world scenarios where SQL is widely used for data management and analysis.

Theory:

DDL Commands:

DDL (Data Definition Language) commands in SQL are used for defining and managing the structure of the database. These commands allow us to create, modify, and delete database objects such as tables, indexes, views, and constraints. Here are some commonly used DDL commands in SQL:

1. **CREATE:** Creates a new database object such as a table, index, view, or schema. For example:
   - **CREATE TABLE:** Creates a new table with specified columns and constraints.

   Syntax:

```
CREATE TABLE table_name
(column1 datatype,
column2 datatype,
column3 datatype,
.....
columnN datatype,
PRIMARY KEY( one or more columns ) );
```

Specifying Basic constraints in SQL:

- The following constraints are commonly used in SQL:
- **NOT NULL** - Ensures that a column cannot have a NULL value
- **UNIQUE** - Ensures that all values in a column are different

- **PRIMARY KEY** - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table
- **FOREIGN KEY** - A foreign key is a column or a set of columns in one table that references the primary key/unique key columns in another table.
- **CHECK** - Ensures that all values in a column satisfies a specific condition
- **DEFAULT** - Sets a default value for a column when no value is specified.

  Example:

  CREATE TABLE STUDENT(

  ROLLNO NUMBER **UNIQUE,**

  USN VARCHAR2(10) **PRIMARY KEY,**

  NAME VARCHAR2(10),

  DEPT VARCHAR2(5) **NOT NULL,**

  AGE NUMBER **CHECK** AGE>19,

  ADDRESS VARCHAR2(10) **DEFAULT** 'GIT');

SPECIFYING FOREIGN KEY CONSTRAINTS:

A FOREIGN KEY is a column (or field) in one table, that refers to the PRIMARY KEY in another table. The table with the foreign key is called the child table, and the table with the primary key is called the referenced or parent table.

The FOREIGN KEY constraint prevents invalid data from being inserted into the foreign key column, because it has to be one of the values contained in the parent table.

Syntax:

FOREIGN KEY (*col_name*, ...) REFERENCES *tbl_name* (*col_name*,...)[ON

DELETE *reference_option*] (optional)

[ON UPDATE *reference_option*](optional)

reference_option: RESTRICT | CASCADE | SET NULL | NO ACTION | SET DEFAULT

Various ways of specifying foreign key constraints,

Example: Suppose **DID** is an attribute in Child table and another **DID** is in Parent (Department) table.

1. FOREIGN KEY (DID) REFERENCES DEPARMENT (DID)
2. FOREIGN KEY (DID) REFERENCES DEPARTMENT (DID) ON **DELETE CASCADE**
3. FOREIGN KEY (DID) REFERENCES DEPARTMENT (DID) ON **DELETE SET NULL**
4. FOREIGN KEY (DID) REFERENCES DEPARTMENT (DID) ON **DELETE SET DEFAULT**
5. FOREIGN KEY (DID) REFERENCES DEPARTMENT (DID) ON **UPDATE CASCADE**
6. FOREIGN KEY (DID) REFERENCES DEPARTMENT (DID) ON **UPDATE SET NULL**
7. FOREIGN KEY (DID) REFERENCES DEPARTMENT (DID) ON **UPDATE SET DEFAULT**

2. **ALTER:** Modifies the structure of an existing database object. Some common ALTER commands are:
    - ALTER TABLE: Modifies the structure of a table, such as adding or dropping columns, altering column properties, or renaming the table.

      Syntax

      ALTER TABLE table_name ADD column_name datatype;

      ALTER TABLE *table_name*

      DROP COLUMN *column_name*;

      ALTER TABLE *table_name*

      RENAME COLUMN *old_name* to *new_name*;

ALTER TABLE *table_name*

MODIFY COLUMN *column_name datatype*;

3. **DROP:** Deletes an existing database object.

   For example:

   - DROP TABLE: Deletes a table and all its associated data.

     DROP TABLE *table_name*;

   - DROP SCHEMA: Deletes a schema and all its objects.

DML Commands:

DML (Data Manipulation Language) commands in SQL are used to manipulate and retrieve data within the database tables. These commands allow us to insert, update, delete, and retrieve data from the tables. Here are some commonly used DML commands in SQL:

1. **SELECT:** Retrieves data from one or more tables based on specified conditions. It is used to query and retrieve data from the database. For example:
   - SELECT * FROM table_name: Retrieves all columns and rows from a table.
   - SELECT column1, column2 FROM table_name: Retrieves specific columns from a table.
2. **INSERT:** Inserts new records (rows) into a table. For example:
   - INSERT INTO table_name (column1, column2) VALUES (value1, value2): Inserts a new row into a table with specified values.
3. **UPDATE:** Modifies existing records in a table. For example:

   UPDATE table_name

   SET column1 = value1, column2 = value2

   WHERE condition;

   Updates the values of specified columns in a table based on a condition.

4. **DELETE:** Deletes records from a table. For example:
   - DELETE FROM table_name WHERE condition;

   Deletes rows from a table based on a condition.

Application of the experiment performed: (TO BE WRITTEN BY STUDENT)

[***PLEASE ASK STUDENTS TO WRITE THE GIVEN PROBLEM DEFINITION]***

Procedure/Algorithm

[STUDENTS ARE SUPPOSED TO WRITE THE STEPS TO CREATE TABLE, INSERTVALUES INTO TABLE AND WRITE QUERIES]

Output and Observation:

[PRINTOUT OF RESULTS OF CREATE STATEMENT, INSERT STATEMENTS AND QUERIES TO BEATTACHED]

Outcome of the Experiment: Students able to implement SQL, which can include a result set displaying retrieved data for SELECT statements, or database changes like inserted, updated, or deleted rows for INSERT, UPDATE, and DELETE statements. Additionally, it can result in structural changes to the database schema or user permissions.

Term work No.6

DDL and DML Commands

Objective of the experiment: SQL experiment is to equip students with the necessary skills and knowledge to effectively interact with and manipulate relational databases using SQL queries. This practical experience helps students gain a deeper understanding of the theoretical concepts covered in the DBMS course and prepares them for real-world scenarios where SQL is widely used for data management and analysis.

Theory:

SQL aggregate functions perform calculations on multiple values and return a single value.

They are often used with the GROUP BY clause to group rows that have the same values in

specified columns. Here are some common SQL aggregate functions:

1. **COUNT()**: Returns the number of rows that match the specified criteria.
    - Example: SELECT COUNT(*) FROM employees;
2. **SUM()**: Returns the total sum of a numeric column.
    - Example: SELECT SUM(salary) FROM employees;
3. **AVG()**: Returns the average value of a numeric column.
    - Example: SELECT AVG(salary) FROM employees;
4. **MIN()**: Returns the minimum value in a column.
    - Example: SELECT MIN(salary) FROM employees;
5. **MAX()**: Returns the maximum value in a column.
    - Example: SELECT MAX(salary) FROM employees;
6. **GROUP_CONCAT()** (in some databases like MySQL): Returns a concatenated string of values from a group.
    - Example: SELECT GROUP_CONCAT(employee_name) FROM employees;

These functions are essential for summarizing and analyzing data in a database.

SQL nested queries, also known as subqueries, are queries within another SQL query. They allow for more complex and powerful data retrieval and manipulation. Here are some common uses and examples of nested queries:

1. Subquery in a SELECT Statement

A subquery can be used in the SELECT clause to retrieve data from another table.

Example:

sql

Copy code

```
SELECT employee_name,
    (SELECT department_name
     FROM departments
     WHERE departments.department_id = employees.department_id) AS department_name
FROM employees;
```

2. Subquery in a WHERE Clause

A subquery can be used in the WHERE clause to filter the results based on the data from another query.

Example:

sql

Copy code

```
SELECT employee_name
```

```
FROM employees
WHERE department_id =
    (SELECT department_id
     FROM departments
     WHERE department_name = 'Sales');
```

3. Subquery in a FROM Clause

A subquery can be used in the FROM clause to create a derived table.

Example:

sql

Copy code

```
SELECT department_name, AVG(salary)
FROM (SELECT department_id, salary
    FROM employees) AS emp
JOIN departments ON emp.department_id = departments.department_id
GROUP BY department_name;
```

4. Subquery with EXISTS

A subquery can be used with the EXISTS keyword to check for the existence of rows in another table.

Example:

sql

Copy code

```
SELECT employee_name
FROM employees
WHERE EXISTS
    (SELECT 1
     FROM departments
     WHERE departments.department_id = employees.department_id
      AND department_name = 'Sales');
```

5. Subquery with IN

A subquery can be used with the IN keyword to filter results based on a set of values returned by the subquery.

Example:

sql

Copy code

```
SELECT employee_name
FROM employees
WHERE department_id IN
    (SELECT department_id
     FROM departments
```

```
    WHERE location_id = 100);
```

6. Correlated Subquery

A correlated subquery is a subquery that references columns from the outer query. It is executed once for each row processed by the outer query.

Example:

sql

Copy code

```
SELECT employee_name, salary

FROM employees e1

WHERE salary >

    (SELECT AVG(salary)

     FROM employees e2

     WHERE e1.department_id = e2.department_id);
```

Nested queries are powerful tools for complex data retrieval and allow for advanced data manipulation and analysis in SQL.

Application of the experiment performed: (TO BE WRITTEN BY STUDENT)

[***PLEASE ASK STUDENTS TO WRITE THE GIVEN PROBLEM DEFINITION]***

Procedure/Algorithm

[STUDENTS ARE SUPPOSED TO WRITE THE STEPS TO CREATE TABLE, INSERTVALUES INTO TABLE AND WRITE QUERIES]

Output and Observation:

[PRINTOUT OF RESULTS OF CREATE STATEMENT, INSERT STATEMENTS AND QUERIES TO BEATTACHED]

Outcome of the Experiment: Students able to implement SQL, which can include a result set displaying retrieved data for SELECT statements, or database changes like inserted, updated, or deleted rows for INSERT, UPDATE, and DELETE statements. Additionally, it can result in structural changes to the database schema or user permissions.

Term work No. 7

PL / SQL

Objective of the Experiment: PL/SQL (Procedural Language/SQL) is an extension of SQL designed to provide procedural capabilities within the Oracle Database environment.

PL/SQL provides basic and advanced concepts of SQL. Our PL/SQL tutorial is designed for beginners and professionals. PL/SQL is a block structured language that can have multiple blocks in it.

Our PL/SQL tutorial includes all topics of PL/SQL language such as conditional statements, loops, arrays, string, exceptions, collections, records, triggers, functions, procedures, cursors etc. There are also given PL/SQL interview questions and quizzes to help you better understand the PL/SQL language.

SQL stands for Structured Query Language i.e. used to perform operations on the records stored in database such as inserting records, updating records, deleting records, creating, modifying anddropping tables, views etc.

What is PL/SQL

PL/SQL is a block structured language. The programs of PL/SQL are logical blocks that can contain any number of nested sub-blocks. Pl/SQL stands for "Procedural Language extension of SQL" that is used in Oracle. PL/SQL is integrated with Oracle database (since version 7). The functionalities of PL/SQL usually extended after each release of Oracle database. Although PL/SQL is closely integrated with SQL language, yet it adds some programming constraints that are not available in SQL.

PL/SQL Functionalities

PL/SQL includes procedural language elements like conditions and loops. It allows declarationof constants and variables, procedures and functions, types and variable of those types and triggers. It can support Array and handle exceptions (runtime errors). After the implementation of version 8 of Oracle database have included features associated with object orientation. You can create PL/SQL units like procedures, functions, packages, types and triggers, etc. which arestored in the database for reuse by applications.

With PL/SQL, you can use SQL statements to manipulate Oracle data and flow of controlstatements to process the data.

The PL/SQL is known for its combination of data manipulating power of SQL with data processing power of procedural languages. It inherits the robustness, security, and portability ofthe Oracle Database.

PL/SQL is not case sensitive so you are free to use lower case letters or upper case letters except within string and character literals. A line of PL/SQL text contains groups of characters known as lexical units. It can be classified as follows:

- De limiters
- Identifiers
- Literals
- Comments

PL/SQL Variables

A variable is a meaningful name which facilitates a programmer to store data temporarily during the execution of code. It helps you to manipulate data in PL/SQL programs. It is nothing except aname given to a storage area. Each variable in the PL/SQL has a specific data type which definesthe size and layout of the variable's memory.

A variable should not exceed 30 characters. Its letter optionally followed by more letters, dollarsigns, numerals, underscore etc.

How to declare variable in PL/SQL

You must declare the PL/SQL variable in the declaration section or in a package as a global variable. After the declaration, PL/SQL allocates memory for the variable's value and the storagelocation is identified by the variable name.

variable_name [CONSTANT] datatype [NOT NULL] [:= | DEFAULT initial_value]

Declaration Restrictions:

In PL/SQL while declaring the variable some restrictions hold.

- Forward references are not allowed i.e. you must declare a constant or variable beforereferencing it in another statement even if it is a declarative statement.
  val number := Total
  - 200;Total number
  := 1000;

  The first declaration is illegal because the TOTAL variable must be declared before using it in anassignment expression.

- Variables belonging to the same datatype cannot be declared in the same statement.N1, N2, N3 Number;
  It is an illegal declaration.

Naming rules for PL/SQL variables

The variable in PL/SQL must follow some naming rules like other programming languages.

- The variable_name should not exceed 30 characters.
- Variable name should not be the same as the table table's column of that block.
- The name of the variable must begin with ASCII letter. The PL/SQL is not case sensitiveso it could be either lowercase or uppercase. For example: v_data and V_DATA refer to the same variables.
- You should make your variable easy to read and understand, after the first character, itmay be any number, underscore (_) or dollar sign ($).
- NOT NULL is an optional specification on the variable.

Initializing Variables in PL/SQL Evertime you declare a variable, PL/SQL defines a default value NULL to it. If you want to initialize a variable with other value than NULL value, you can do so during the declaration, by using any one of the following methods.

You can also specify NOT NULL constraint to avoid NULL value. If you specify the NOT NULL constraint, you must assign an initial value for that variable.

You must have a good programming skill to initialize variable properly otherwise, sometimesprogram would produce unexpected result.

Variable Scope in PL/SQL:

PL/SQL allows nesting of blocks. A program block can contain another inner block. If youdeclare a variable within an inner block, it is not accessible to an outer block.

Application of the experiment performed: (TO BE WRITTEN BY STUDENT)

[***PLEASE ASK STUDENTS TO WRITE THE GIVEN PROBLEM DEFINITION]***

Procedure/Algorithm

[STUDENTS ARE SUPPOSED TO WRITE THE STEPS TO CREATE TABLE, INSERTVALUES INTO TABLE AND WRITE QUERIES]

Output and Observation:

[PRINTOUT OF RESULTS OF CREATE STATEMENT, INSERT STATEMENTS AND QUERIES TO BE ATTACHED]

Outcome of the Experiment: Students able to implement PL/SQL language to design different Database applications.

Term work No. 8

PL / SQL

Objective of the Experiment: PL/SQL (Procedural Language/SQL) is an extension of SQL designed to provide procedural capabilities within the Oracle Database environment.

In PL/SQL, loops are used to execute a sequence of statements multiple times. The two main types of loops are the `WHILE` loop and the `FOR` loop. Here are explanations and examples for both:

WHILE Loop

The `WHILE` loop repeatedly executes a block of statements as long as a specified condition is true.

Syntax:

```
sql
Copy code
WHILE condition LOOP
   -- statements
END LOOP;
```

Example:

```
sql
Copy code
DECLARE
   counter NUMBER := 1;
BEGIN
   WHILE counter <= 5 LOOP
      DBMS_OUTPUT.PUT_LINE('Counter value is: ' || counter);
      counter := counter + 1;
   END LOOP;
END;
/
```

In this example, the `WHILE` loop will print the value of `counter` from 1 to 5.

FOR Loop

The `FOR` loop executes a block of statements a specific number of times, typically using a range of values.

Syntax:

```
sql
Copy code
FOR loop_variable IN lower_bound..upper_bound LOOP
   -- statements
END LOOP;
```

Example:

```
sql
Copy code
BEGIN
   FOR counter IN 1..5 LOOP
      DBMS_OUTPUT.PUT_LINE('Counter value is: ' || counter);
   END LOOP;
END;
/
```

In this example, the `FOR` loop will automatically iterate from 1 to 5, printing the value of `counter` for each iteration.

Reverse FOR Loop

The `FOR` loop can also iterate in reverse order by using the `REVERSE` keyword.

Syntax:

```
sql
Copy code
FOR loop_variable IN REVERSE lower_bound..upper_bound LOOP
   -- statements
END LOOP;
```

Example:

```
sql
Copy code
BEGIN
   FOR counter IN REVERSE 5..1 LOOP
      DBMS_OUTPUT.PUT_LINE('Counter value is: ' || counter);
   END LOOP;
END;
/
```

In this example, the `FOR` loop will iterate from 5 to 1 in reverse order, printing the value of `counter` for each iteration.

Both `WHILE` and `FOR` loops are powerful constructs in PL/SQL for controlling the flow of execution and repeating tasks as needed.

Application of the experiment performed: (TO BE WRITTEN BY STUDENT)

[***PLEASE ASK STUDENTS TO WRITE THE GIVEN PROBLEM DEFINITION]***

Procedure/Algorithm

[STUDENTS ARE SUPPOSED TO WRITE THE STEPS TO CREATE TABLE, INSERTVALUES INTO TABLE AND WRITE QUERIES]

Output and Observation:

[PRINTOUT OF RESULTS OF CREATE STATEMENT, INSERT STATEMENTS AND QUERIES TO BE ATTACHED]

Outcome of the Experiment: Students able to implement PL/SQL language to design different Database applications.