Institute's Vision

To be one of the leading Institutions for Engineering education developing proficient Engineers with global acceptance in the service of mankind.

Institute's Mission

1. Providing quality Engineering education to cater the needs of Industry and society with multidisciplinary approach on sustainable basis.

2. Developing globally competent Engineers having ability to solve real-life problems addressing environmental issues through technological advancements.

3. Inculcating professionalism, teamwork, research, innovation and entrepreneurship, maintaining the spirit of continuous learning.

4. Fostering the collaboration with Industry, academia, research organizations, experts and alumni.

5. Imparting employability skills, nurturing leadership qualities with ethical and social values among students.

Department’s Vision

To be one of the leading Departments for Computer Science & Engineering education, developing proficient Engineers with global acceptance in the service of mankind.

Department’s Mission

1. Providing technical skills with strong fundamentals of Computer Science discipline with an emphasis on software development.

2. Inculcating analytical, programming and multidisciplinary skills to enhance employability.

3. Fostering problem-solving, team-building, and lifelong learning skills with societal, environmental and ethical sense.

4. Developing researchers and entrepreneurs to solve real-life problems through industry interactions and collaborations.

Program Educational Objectives (PEOs)

Graduates of Computer Science & Engineering employed should be able to

1. Analyze Computer Science & Engineering techniques, relate them with real life problems and provide solutions that are technically sound, economically viable and socially acceptable.

2. Utilize acquired programming, analytical, design and implementation skills to formulate and solve computational problems.

3. Evolve as competent professionals, researchers and entrepreneurs having collaborative and leadership skills with effective communication abilities to pursue appropriate career options and become capable of working in multi-disciplinary environment.

4. Excel as socially committed Computer Engineers having good human and ethical values.

Program Specific Outcomes (PSOs)

PSOs are a statement that describes what students are expected to know and be able to do in a specialized area of discipline upon graduation from a program.

PSO1: Apply knowledge of core courses and emerging areas including Data Science, AI/ML, Cloud Computing, Information security, Image Processing for solving real life problems.

PSO2: Design and develop software and hardware systems using latest technologies, programming languages, and open-source platforms.

PSO3: Apply standard software engineering principles and professional skills to create solutions that meet Industry needs.

Program Outcomes (POs)

POs are statements that describe what students are expected to know and be able to do upon graduating from the program. These relate to the skills, knowledge, analytical ability attitude and behaviour that students acquire through the program.

PO1-Engineering Knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2-Problem Analysis: Identify, formulate, review research literature, and analyse complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences and engineering sciences.

PO3-Design/Development of Solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO4-Conduct Investigations of Complex Problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions for complex problems

PO5-Modern Tool Usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

PO6-The Engineer and Society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO7-Environment and Sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8-Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9-Individual and Team Work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10-Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO11-Project Management and Finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO12-Life-long Learning: Recognize the need for, and have the preparation and ability to engage in independent and lifelong learning in the broadest context of technological change.

Course Outcomes:

At the end of the course students will be able to:

CO1: Demonstrate static and dynamic memory allocation techniques.

CO2: Implement various linear and non-linear data structures.

CO3: Develop for searching and sorting techniques.

CO4: Analyze different data structures in terms of time and space complexities.

CO – PO Mapping

PO's→ CO's↓	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO 1	3	2			2							3
CO 2	3	2	2	2	2							3
CO 3	3	2			2							3
CO 4	3	2	2	2	2				2	2	2	3

CO-PSO Mapping

PSO's → CO's↓	PSO1	PSO2	PSO3
CO 1	3		
CO 2	2		
CO 3	2		
CO 4	3	2	2

Continuous Evaluation Rubrics

Criterion	Excellent (5-6)	Satisfactory (3-4)	Poor (2-3)
Understanding of Problem statement	Demonstrates thorough understanding of concepts, confidently answering all questions.	Demonstrates basic understanding of concepts but struggles to answer few questions.	Demonstrates limited understanding of concepts and struggles to answer basic questions.
SQL Query Accuracy	All queries are correct, optimized, and meet the requirements	Most queries are correct; minor optimization issues..	Queries work, but with minor errors or inefficiencies.
Code Organization & Error Handling	Code is clean, modular, and well-structured with comments and Comprehensive handling of errors and exceptions	Code is mostly organized and readable; minor commenting issues. Basic error handling implemented.	Code is somewhat organized; limited comments and struggling with error handling
Documentation (Journal writing)	Thorough documentation with all key aspects explained.	Adequate documentation with most key aspects explained.	Minimal documentation with unclear information.
Timely Completion of Task	Completes tasks within The Alotted time	Completes tasks withminor delays	Rarely completes taskson time.

Do’s and Don’ts for Students in the Computer Lab

Do’s

- Be punctual and adhere to the lab schedule.
- Ensure that you record entry and exit times in the log register.
- Use only the computer assigned to you unless instructed otherwise.
- Save your work regularly and back it up to an external device or cloud storage.
- Use the internet responsibly and for academic purposes only.
- Only print what is necessary. Be mindful of paper and ink usage.
- Handle all equipment with care.
- Report any malfunctions or damage to the lab supervisor immediately.
- Maintain silence and avoid any behavior that may disturb others.
- Log out from your account and shut down the computer properly after use.
- Ensure that your workspace is clean and tidy before you leave.
- Arrange the chairs properly before leaving the lab.

Don’ts

- Do not plagiarize (copy) others' work. Maintain academic integrity.
- Do not access, modify, or delete files that belong to others.
- Do not access inappropriate websites or engage in any form of cyber misconduct.
- Do not download or install unauthorized software.
- Do not attempt to bypass network security or engage in hacking activities.
- Do not remove, alter, or tamper with any hardware or software configurations.
- Do not use mobile phone during lab sessions.
- Do not consume food or drinks inside the lab.

By following these guidelines, students can help maintain a productive and respectful environment in the computer lab

List of Experiments

PRACTICAL NO. 1
Study of Codd's Rules in Database System

- □ **Aim**:
  To Study the 12 Codd's rules in database system.

- □ **Theory**:

Rule 1: Information Rule

The data stored in a database, may it be user data or metadata, must be a value of some table cell. Everything in a database must be stored in a table format.

Rule 2: Guaranteed Access Rule

Every single data element (value) is guaranteed to be accessible logically with a combination of table-name, primary-key (row value), and attribute-name (column value). No other means, such as pointers, can be used to access data.

Rule 3: Systematic Treatment of NULL Values

The NULL values in a database must be given a systematic and uniform treatment. This is a very important rule because a NULL can be interpreted as one the following − data is missing, data is not known, or data is not applicable.

Rule 4: Active Online Catalog

The structure description of the entire database must be stored in an online catalog, known as data dictionary, which can be accessed by authorized users. Users can use the same query language to access the catalog which they use to access the database itself.

Rule 5: Comprehensive Data Sub-Language Rule

A database can only be accessed using a language having linear syntax that supports data definition, data manipulation, and transaction management operations. This language can be useddirectly or by means of some application. If the database allows access to data without any help of this language, then it is considered as a violation.

Rule 6: View Updating Rule

All the views of a database, which can theoretically be updated, must also be updatable by the system.

Rule 7: High-Level Insert, Update, and Delete Rule

A database must support high-level insertion, updation, and deletion. This must not be limited to a single row, that is, it must also support union, intersection and minus operations to yield sets of data records.

Rule 8: Physical Data Independence

The data stored in a database must be independent of the applications that access the database. Any change in the physical structure of a database must not have any impact on how the data is being accessed by external applications.

Rule 9: Logical Data Independence

The logical data in a database must be independent of its user's view (application). Any changein logical data must not affect the applications using it. For example, if two tables are merged or one is split into two different tables, there should be no impact or change on the user application. This isone of the most difficult rule to apply.

Rule 10: Integrity Independence

A database must be independent of the application that uses it. All its integrity constraints can be independently modified without the need of any change in the application. This rule makes a database independent of the front-end application and its interface.

Rule 11: Distribution Independence

The end-user must not be able to see that the data is distributed over various locations. Users should always get the impression that the data is located at one site only. This rule has been regarded as the foundation of distributed database systems.

Rule 12: Non-Subversion Rule

If a system has an interface that provides access to low-level records, then the interface must not be able to subvert the system and bypass security and integrity constraints.

□ **Conclusion**

In conclusion, the Study of Codd's rules highlights the fundamental principles of relational databases ensuring data consistency, integrity, and independence, which are essential for building reliable and efficient database Management System.

PRACTICAL NO. 2
Study and Design of Entity-Relationship Model Diagram

- ❑ **Aim**:
  Study and Design of Entity-Relationship Model Diagram

- ❑ **Theory**:

Entity-Relationship (E-R) model concepts
– entities
– relationships
– cardinality constraints
– entity type hierarchies
– weak entities

Entity-Relationship model is used in the conceptual design of a database (☞conceptual level, conceptual schema)

• **Design is independent of all physical considerations (DBMS, OS, . . .)**

Questions that are addressed during conceptual design:

– What are the entities and relationships of interest (mini world)?
– What information about entities and relationships among entities needs to be stored in the database?
– What are the constraints (or business rules) that (must) hold for the entities and relationships?

A database schema in the ER model can be represented pictorially(Entity-Relationship diagram)

Entity: real-world object or thing with an independent existence and which is distinguishable from other objects. Examples are a person, car, customer, product, gene, book etc.

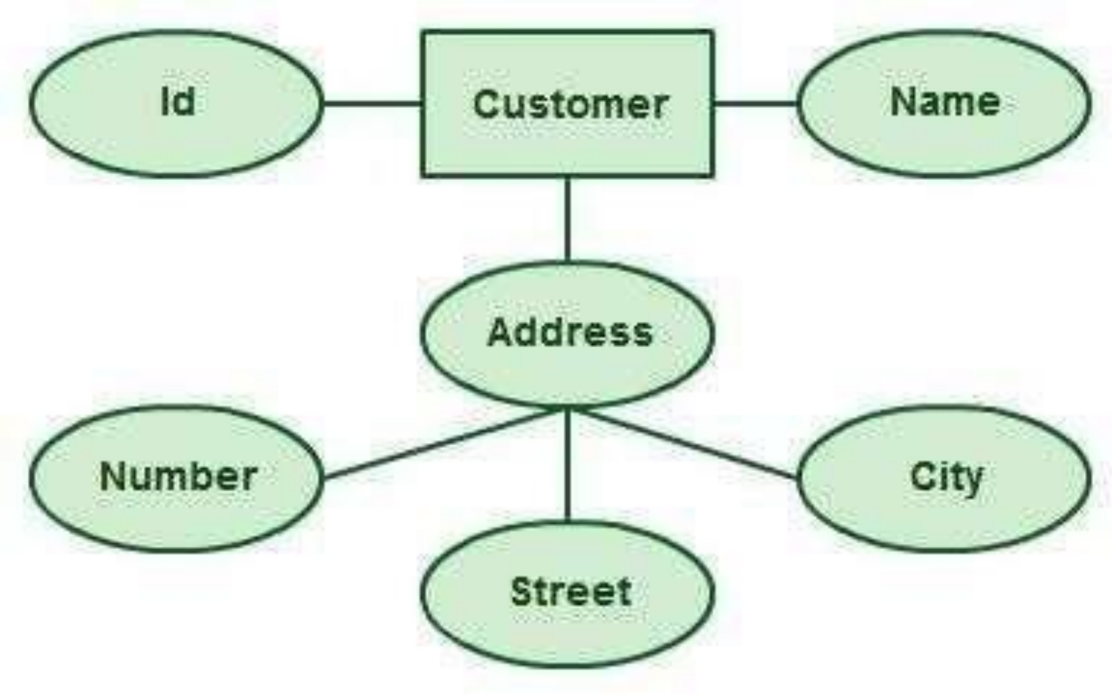
• **Attributes:** an entity is represented by a set of attributes (its descriptive properties), e.g., name, age, salary, price etc. Attribute values that describe each entity become a major part of the data eventually stored in a database. With each attribute a domain is associated, i.e., a set of permitted values for an attribute. Possible domains are integer, string, date, etc

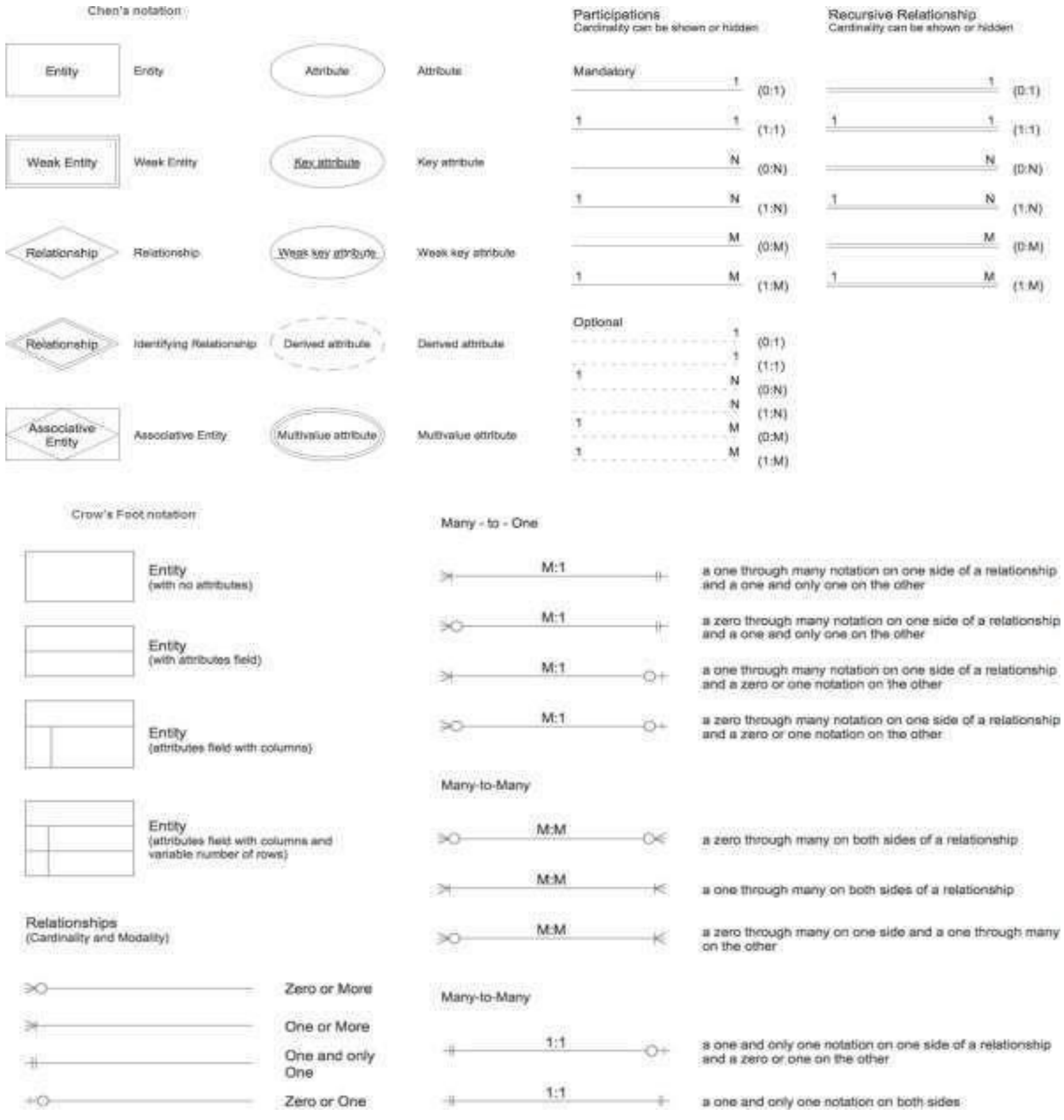
- **Entity Type:** Collection of entities that all have the same attributes, e.g., persons, cars, customers etc.

- **Entity Set:** Collection of entities of a particular entity type at any point in time; entity set is typically referred to using the same name as entity type.

Key attributes of an Entity Type

- Entities of an entity type need to be distinguishable.
- A superkey of an entity type is a set of one or more attributes whose values uniquely determine each entity in an entity set.
- A candidate key of an entity type is a minimal (in terms of number of attributes) superkey.
- For an entity type, several candidate keys may exist. During conceptual design, one of the candidate keys is selected to be the primary key of the entity type.




Chen's notation
Entity
Entity
Attribute
Attribute
Weak Entity
Weak Entity
Key attribute
Key attribute
Relationship
Relationship
Weak key attribute
Weak key attribute
Relationship
Identifying Relationship
Derived attribute
Derived attribute
Associative Entity
Associative Entity
Multivalue attribute
Multivalue attribute
Participations
Cardinality can be shown or hidden
Mandatory
Optional
(0:1)
(1:1)
(0:N)
(1:N)
(0:M)
(1:M)
1
N
M
Recursive Relationship
Cardinality can be shown or hidden
Crow's Foot notation
Entity
(with no attributes)
Entity
(with attributes field)
Entity
(attributes field with columns)
Entity
(attributes field with columns and variable number of rows)
Relationships
(Cardinality and Modality)
Zero or More
One or More
One and only One
Zero or One
Many - to - One
M:1
a one through many notation on one side of a relationship and a one and only one on the other
a zero through many notation on one side of a relationship and a one and only one on the other
a one through many notation on one side of a relationship and a zero or one notation on the other
a zero through many notation on one side of a relationship and a zero or one notation on the other
Many-to-Many
M:M
a zero through many on both sides of a relationship
a one through many on both sides of a relationship
a zero through many on one side and a one through many on the other
Many-to-Many
1:1
a one and only one notation on one side of a relationship and a zero or one on the other
a one and only one notation on both sides

□

□

□

□

□

□

□ Conclusion

The design of E-R diagrams is indispensable for effective database management and development. It not Only enhances understanding and communication but also lays the groundwork for a well-structured database system that meets the needs of its users.

PRACTICAL NO. 3
Implementation of DDL& DML Commands of SQL with Suitable Examples

□ **Aim:**

The aim of this document is to provide a comprehensive understanding of DDL and DML commands in SQL, demonstrating their usage through examples, and highlighting their importance in database management.

□ **Theory**:

DDL is concerned with the structure of the database, allowing the creation, alteration, and deletion of database objects like tables, indexes, and schemas. Key DDL commands include CREATE, ALTER, DROP, and TRUNCATE.

DML is focused on managing the data within these structures. It allows users to insert, update, delete, and retrieve data. Key DML commands include INSERT, SELECT, UPDATE, and DELETE.

Implementation with Examples

DDL Commands

1. **CREATE**: Creating a new table.

```
CREATE TABLE Employees (
   EmployeeID INT PRIMARY KEY,
   FirstName VARCHAR(50),
   LastName VARCHAR(50),
   HireDate DATE,
   Salary DECIMAL(10, 2)
);
```

This command creates an Employees *table with fields for employee ID, first name, last name, hire date, and salary.*

2. **ALTER**: Modifying an existing table.

```
ALTER TABLE Employees
ADD Email VARCHAR(100);
```

This command adds an Email *column to the existing* Employees *table.*

3. **DROP**: Deleting a table.

```
DROP TABLE Employees;
```

This command removes the Employees *table and all its data from the database.*

4. **TRUNCATE**: Removing all records from a table.

```
TRUNCATE TABLE Employees;
```

This command deletes all records in the Employees *table without removing the table itself.*

DML Commands

1. **INSERT**: Adding a new record.

```
INSERT INTO Employees (EmployeeID, FirstName, LastName, HireDate, Salary)
VALUES (1, 'Alice', 'Smith', '2023-01-10', 70000.00);
```

This command inserts a new employee record into the Employees *table.*

2. **SELECT**: Retrieving data.

```
SELECT * FROM Employees;
```

This command retrieves all records from the Employees *table.*

Filtered selection:

```
SELECT FirstName, LastName FROM Employees WHERE Salary > 60000;
```

This command retrieves the names of employees earning more than $60,000.

3. **UPDATE**: Modifying an existing record.

```
UPDATE Employees
SET Salary = Salary * 1.05
WHERE EmployeeID = 1;
```

This command updates the salary of the employee with EmployeeID *1, increasing it by 5%.*

4. **DELETE**: Removing a record.

```
DELETE FROM Employees
WHERE EmployeeID = 1;
```

This command deletes the employee record with EmployeeID *1 from the* Employees *table.*

Conclusion

DDL and DML commands form the backbone of SQL, allowing users to define the database structure and manipulate the data it contains. Understanding how to use these commands effectively is essential for database management and operations.

- **DDL** commands enable the creation and maintenance of database schemas, which define how data is organized.
- **DML** commands facilitate interaction with the data, making it possible to add, modify, and retrieve information.

PRACTICAL NO. 4
Applying Integrity Constraints on the Table

- □ **Aim**:

To enforce data accuracy and consistency through integrity constraints.

- □ **Theory**:

SQL Create Constraints

Constraints can be specified when the table is created with the CREATE TABLE statement, or after the table is created with the ALTER TABLE statement.

Syntax

```
CREATE TABLE table_name (
   column1 datatype constraint,
   column2 datatype constraint,
   column3 datatype constraint,
   ....
);
```

Constraints are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the table. If there is any violation between the constraint and the data action, the action is aborted.

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

The following constraints are commonly used in SQL:

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

```
create table second (id int ,name char(10) not null,pass int unique,primary key(id))
create table second (id int ,name char(10),pass int ,primary key(id),unique(name));

create table third (id int, roll int,foreign key (id) reference second(id));
```

- **Conclusion**

This enhances the overall reliability of the database system.

PRACTICAL NO. 5
Implementation of Different Clauses in SQL

□ **Aim**:
To utilize various SQL clauses for efficient data manipulation and retrieval.

□ **Theory**:

The SQL **AND** & **OR** operators are used to combine multiple conditions to narrow data in an SQL statement. These two operators are called as the conjunctive operators.

These operators provide a means to make multiple comparisons with different operators in the same SQL statement.

The AND Operator

The **AND** operator allows the existence of multiple conditions in an SQL statement's WHERE clause.

Syntax

The basic syntax of the AND operator with a WHERE clause is as follows −

```
SELECT column1, column2, columnN
FROM table_name
WHERE [condition1] AND [condition2]...AND [conditionN];



SQL> SELECT ID, NAME, SALARY

FROM CUSTOMERS

WHERE SALARY > 2000 AND age < 25;
```

The OR Operator

The OR operator is used to combine multiple conditions in an SQL statement's WHERE clause.

Syntax

The basic syntax of the OR operator with a WHERE clause is as follows −

```
SELECT column1, column2, columnN
FROM table_name
WHERE [condition1] OR [condition2]...OR [conditionN]
```

```
SQL> SELECT ID, NAME, SALARY

FROM CUSTOMERS

WHERE SALARY > 2000 OR age < 25;
```

The SQL **LIKE** clause is used to compare a value to similar values using wild card operators. There are two wildcards used in conjunction with the LIKE operator.

□ The percent sign (%)

□ The underscore (_)

The percent sign represents zero, one or multiple characters. The underscore represents a single number or character. These symbols can be used in combinations.

Syntax

The basic syntax of % and _ is as follows −

```
SELECT FROM table_name WHERE
column LIKE 'XXXX%'

or
```

Sr. No.	Statement & Description
1	WHERE SALARY LIKE '200%' Finds any values that start with 200
2	WHERE SALARY LIKE '%200%’ Finds any values that have 200 in any position
3	WHERE SALARY LIKE '_00%' Finds any values that have 00 in the second and third positions
4	WHERE SALARY LIKE '2_%_%' Finds any values that start with 2 and are at least 3 characters in length
5	WHERE SALARY LIKE '%2' Finds any values that end with 2
6	WHERE SALARY LIKE '_2%3' Finds any values that have a 2 in the second position and end with a 3
7	WHERE SALARY LIKE '2_3' Finds any values in a five-digit number that start with 2 and end with 3

```
SQL> SELECT * FROM CUSTOMERSWHERE SALARY LIKE '200%';
```

The SQL **TOP** clause is used to fetch a TOP N number or X percent records from a table.

Note − All the databases do not support the TOP clause. For example MySQL supports the **LIMIT** clause to fetch limited number of records while Oracle uses the **ROWNUM** command to fetch a limited number of records.

Syntax

The basic syntax of the TOP clause with a SELECT statement would be as follows.

```
SELECT TOP number percent column_name(s)FROM
table_name
WHERE [condition]

SQL> SELECT TOP 3* FROM CUSTOMERS;


SQL> SELECT * FROM CUSTOMERSLIMIT3;



SQL> SELECT * FROM CUSTOMERS

WHERE ROWNUM <=3;
```

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

Syntax

The basic syntax of the ORDER BY clause is as follows −

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


SQL> SELECT * FROM CUSTOMERS

   ORDER BY NAME,  SALARY;

SQL> SELECT * FROM CUSTOMERS

   ORDER BY NAME DESC;
```

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

Syntax

The basic syntax of a GROUP BY clause is shown in the following code block. The GROUP BY clause must follow the conditions in the WHERE clause and must precede the ORDER BY clause if one is used.

```
SELECT column1, column2
FROM table_name WHERE [
conditions ]
GROUP BY column1, column2
ORDER BY column1, column2



SQL> SELECT NAME, SUM(SALARY) FROM CUSTOMERSGROUP

  BY NAME
```

The SQL **DISTINCT** keyword is used in conjunction with the SELECT statement to eliminate all the duplicate records and fetching only unique records.

There may be a situation when you have multiple duplicate records in a table. While fetching such records, it makes more sense to fetch only those unique records instead of fetchingduplicate records.

Syntax

The basic syntax of DISTINCT keyword to eliminate the duplicate records is as follows –

```
SELECT DISTINCT column1, column2,.................columnN

FROM table_name

WHERE  [condition]

SQL> SELECT SALARY FROM CUSTOMERS ORDERBY SALARY;
```

Conclusion

This optimizes query performance and improves data management

PRACTICAL NO. 6

View Data: Write and Execute SQLQueries forWorking with Views

- **Aim**:
  To create and manage views for simplified data access and enhanced security.

- **Theory:**

A view is nothing more than a SQL statement that is stored in the database with an associatedname. A view is actually a composition of a table in the form of a predefined SQL query.

A view can contain all rows of a table or select rows from a table. A view can be created fromone or many tables which depends on the written SQL query to create a view.

Views, which are a type of virtual tables allow users to do the following −

- □ Structure data in a way that users or classes of users find natural or intuitive.
- □ Restrict access to the data in such a way that a user can see and (sometimes)modifyexactly what they need and no more.
- □ Summarize data from various tables which can be used to generate reports. Creating

Views

Database views are created using the **CREATE VIEW** statement. Views can be created froma single table, multiple tables or another view.

To create a view, a user must have the appropriate system privilege according to the specificimplementation.

View Data : Write and Execute SQL Queries forWorking with Views

- □ **Aim**:
  To create and manage views for simplified data access and enhanced security.

- □ **Theory**:
  A view is nothing more than a SQL statement that is stored in the database with an associated name. A view is actually a composition of a table in the form of a predefined SQL query.

  A view can contain all rows of a table or select rows from a table. A view can be created from one or many tables which depends on the written SQL query to create a view.

  Views, which are a type of virtual tables allow users to do the following −

  - □Structure data in a way that users or classes of users find natural or intuitive.
  - □Restrict access to the data in such a way that a user can see and (sometimes)modify exactly what they need and no more.
  - □Summarize data from various tables which can be used to generate reports. Creating

Views

Database views are created using the **CREATE VIEW** statement. Views can be created from a single table, multiple tables or another view.

To create a view, a user must have the appropriate system privilege according to the specific implementation.

The basic **CREATE VIEW** syntax is as follows −

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



SQL > CREATE VIEW CUSTOMERS_VIEW AS
SELECT name, age FROM
CUSTOMERS;



SQL > SELECT * FROM CUSTOMERS_VIEW;
```

The WITH CHECK OPTION

The WITH CHECK OPTION is a CREATE VIEW statement option. The purpose of the WITH CHECK OPTION is to ensure that all UPDATE and INSERTs satisfy the condition(s) in the view definition.

If they do not satisfy the condition(s), the UPDATE or INSERT returns an error.

The following code block has an example of creating same view CUSTOMERS_VIEW with the WITH CHECK OPTION.

```
CREATE VIEW CUSTOMERS_VIEW AS
SELECT name, age
FROM CUSTOMERS
WHERE age IS NOT NULL
WITH CHECK OPTION;
```

The WITH CHECK OPTION in this case should deny the entry of any NULL values in the view's AGE column, because the view is defined by data that does not have a NULL value in the AGE column.

Updating a View

A view can be updated under certain conditions which are given below −

□The SELECT clause may not contain the keyword DISTINCT.

□The SELECT clause may not contain summary functions.

□The SELECT clause may not contain set functions.

□The SELECT clause may not contain set operators.

□The SELECT clause may not contain an ORDER BY clause.

□The FROM clause may not contain multiple tables.

□The WHERE clause may not contain subqueries.

□The query may not contain GROUP BY or HAVING.

□Calculated columns may not be updated.

□All NOT NULL columns from the base table must be included in the view in order for the INSERT query to function.

So, if a view satisfies all the above-mentioned rules then you can update that view. The following code block has an example to update the age of Ramesh.

```
SQL > UPDATE CUSTOMERS_VIEWSET
  AGE =35
  WHERE name ='Ramesh';
```

Inserting Rows into a View

Rows of data can be inserted into a view. The same rules that apply to the UPDATE command also apply to the INSERT command.

Here, we cannot insert rows in the CUSTOMERS_VIEW because we have not included all the NOT NULL columns in this view, otherwise you can insert rows in a view in a similar way as you insert them in a table.

Deleting Rows into a View

Rows of data can be deleted from a view. The same rules that apply to the UPDATE and INSERT commands apply to the DELETE command.

Following is an example to delete a record having AGE = 22.

```
SQL > DELETE FROM CUSTOMERS_VIEW

  WHERE age =22;
```

Dropping Views

Obviously, where you have a view, you need a way to drop the view if it is no longer needed. The syntax is very simple and is given below −

```
DROP VIEW view_name;
```

Following is an example to drop the CUSTOMERS_VIEW from the CUSTOMERS table.

```
DROP VIEW CUSTOMERS_VIEW;
```

Conclusion

This enables more efficient data retrieval while protecting underlying table structures.

PRACTICAL NO. 7

Implementation of Aggregate Functions and Set Operationsin SQL

□ **Aim**:

To utilize aggregate functions and set operations for comprehensive data analysis and comparison.

□ **Theory**:

SQL has numerous predefined aggregate functions that can be used to write queries to produce exactly this kind of information. The GROUP BY clause specifies how to group rows from a data table when aggregating information, while the HAVING clause filters out rows that do not belong in specified groups.

Aggregate functions perform a variety of actions such as counting all the rows in a table, averaging a column's data, and summing numeric data. Aggregates can also search a table to find the highest "MAX" or lowest "MIN" values in a column. As with other types of queries, you can restrict, or filter out the rows these functions act on with the WHERE clause. For example, if a manager needs to know how many employees work in an organization, the aggregate function named COUNT(*) can be used to produce this information.The COUNT(*) function shown in the below SELECT statement counts all rows in a table.

```
SELECT COUNT(*)
FROM employees;

 COUNT(*)
-----------
24
```

Some of the commonly used aggregate functions are as below -

```
SUM([ALL |DISTINCT] expression )

AVG([ALL |DISTINCT] expression )

COUNT([ALL |DISTINCT] expression )

COUNT(*)
```

```
MAX(expression)

MIN(expression)
```

The ALL and DISTINCT keywords are optional, and perform as they do with the SELECT clauses that you have learned to write. The ALL keyword is the default where the option is allowed. The expression listed in the syntax can be a constant, a function, or any combination of column names, constants, and functions connected by arithmetic operators. However, aggregatefunctions are most often used with a column name. Except the COUNT function, all the aggregate functions do not consider NULL values.

There are two rules that you must understand and follow when using aggregates:

□ Aggregate functions can be used in both the SELECT and HAVING clauses (theHAVING clause is covered later in this chapter).

□Aggregate functions cannot be used in a WHERE clause. Its violation will produce the Oracle ORA-00934 group function is not allowed here error message.

Illustrations

The below SELECT query counts the number of employees in the organization.

```
SELECT COUNT(*)Count

FROM employees;

COUNT

------

24
```

The below SELECT query returns the average of the salaries of employees in the organization.

```
SELECT AVG(Salary) average_sal

FROM employees;
```

```

AVERAGE_SAL

-------------

15694
```

The below SELECT query returns the sum of the salaries of employees in the organization.

```
SELECT SUM(Salary) total_sal
FROM employees;


TOTAL_SAL
----------
87472
```

The below SELECT query returns the oldest and latest hired dates of employees in the organization.

```
SELECT MIN (hire_date) oldest, MAX (hire_date) latest
FROM employees;


OLDEST          LATEST
----------    -------------
16-JAN-83       01-JUL-2012
```

Conclusion

SQL aggregate functions and set operations are vital for effective data summarization and insightful analysis.

PRACTICAL NO. 8

Design SQL Queries for all types of Joins and Sub-Query

□ **Aim**:

To design SQL queries demonstrating all types of joins (INNER, LEFT, RIGHT, FULL) and sub-queries for effective data retrieval and relationship management.

□ **Theory**:

A JOIN clause is used to combine rows from two or more tables, based on a related columnbetween them.

Let's look at a selection from the "Orders" table:

Order ID	Customer ID	Order Date
10308	2	1996-09-18
10309	37	1996-09-19
10310	77	1996-09-20

Then, look at a selection from the "Customers" table:

Customer ID	Customer Name	Contact Name
1	Alfreds Futterkiste	Maria Anders
2	Ana Trujillo Emparedados y helados	Ana Trujillo
3	Antonio Moreno Taquería	Antonio Moreno

Example

```
SELECT Orders.OrderID, Customers.CustomerName,

Orders.OrderDateFROM Orders

INNER JOIN Customers ON Orders.CustomerID=Customers.CustomerID;
```

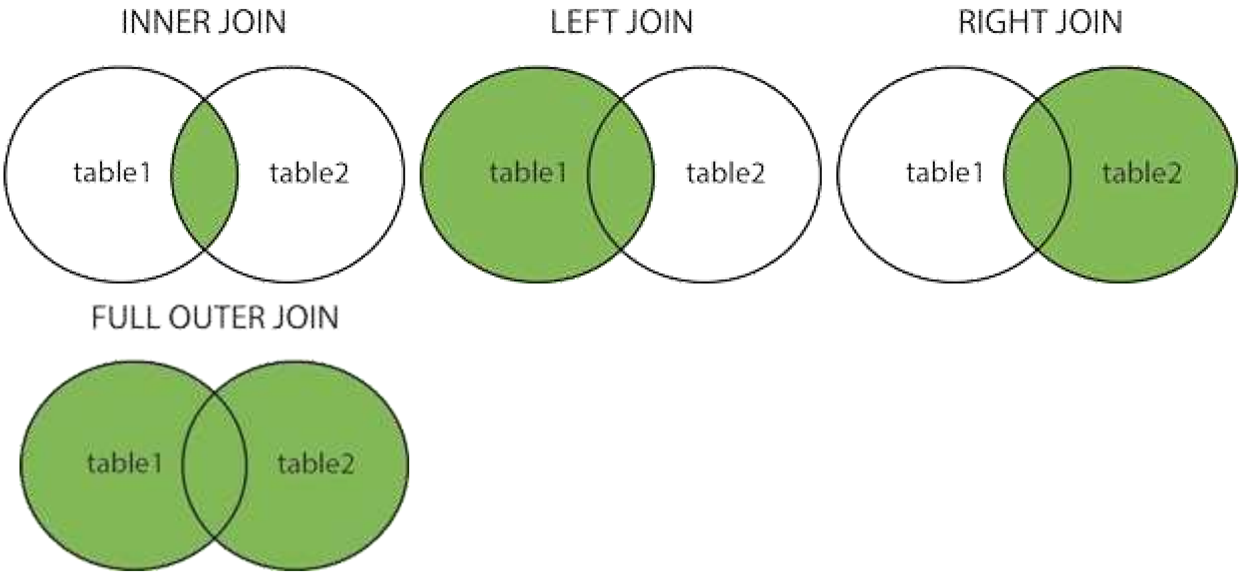
and it will produce something like this:

OrderID	CustomerName
10308	Ana Trujillo Emparedados y helados
10365	Antonio Moreno Taquería
10383	Around the Horn
10355	Around the Horn
10278	Berglunds snabbköp

Different Types of SQL JOINs

Here are the different types of the JOINs in SQL:

- **(INNER) JOIN**: Returns records that have matching values in both tables
- **LEFT (OUTER) JOIN**: Return all records from the left table, and the matched records from the right table
- **RIGHT (OUTER) JOIN**: Return all records from the right table, and the matched records from the left table
- **FULL (OUTER) JOIN**: Return all records when there is a match in either left or right table



INNER Join or EQUI Join

This is a simple JOIN in which the result is based on matched data as per the equality condition specified in the query.

Inner Join Syntax is,

SELECT column-name-list

from table-name1

INNER JOIN

table-name2

WHERE table-name1.column-name = table-name2.column-name;

Natural JOIN

Natural Join is a type of Inner join which is based on column having same name and samedata type present in both the tables to be joined.

Natural Join Syntax is,

SELECT *

from table-name1

NATURAL JOIN

table-name2;

Left Outer Join

The left outer join returns a result table with the matched data of two tables then remaining rows of the left table and null for the right table's column.

Left Outer Join syntax is,

```
SELECT
column-name-
listfrom table-
name1
LEFT OUTER JOIN
table-name2
on table-name1.column-name = table-name2.column-name;
```

Right Outer Join

The right outer join returns a result table with the matched data of two tables then remainingrows of the right table and null for the left table's columns.

Right Outer Join Syntax is,

```
select column-name-list
from table-name1
RIGHT OUTER JOIN
table-name2
on table-name1.column-
name = table-
name2.column-name;
```

Right outer Join Syntax
for Oracle is,

```
select column-name-listfrom
table-name1, table-name2
on table-name1.column-
name(+) = table
name2.column-name;
```

- **Conclusion**

Understanding and implementing various SQL joins and sub-queries is essential for efficiently accessingand combining data across multiple tables, enhancing overall query performance and analytical insights.

PRACTICAL NO. 9

Write and Implement PL/SQL to Retrieve Information from the Databases

□ **Aim**:

To retrieve and display employee information from an employees table in a databaseusing PL/SQL.

□ **Theory**:

PL/SQL (Procedural Language/Structured Query Language) is Oracle Corporation's procedural extension for SQL. It combines the data manipulation power of SQL with the procedural capabilities of programming languages. This allows developers to write complex scripts and functions that can process data and implement business logic directly in the database.

PL/SQL is built around several key components:

- **Blocks**: PL/SQL code is organized into blocks, which are the fundamental units of execution. A block can be anonymous or named (e.g., procedures and functions). Each block consists of three sections:
  - **Declaration**: This section is optional and used to declare variables, constants, cursors, and exceptions.
  - **Execution**: This is the mandatory section where the actual SQL and PL/SQL statements are executed.
  - **Exception Handling**: This optional section deals with errors and exceptions that may occur during execution.

□ **Variables and Data Types**: PL/SQL supports various data types, including scalar types (e.g., NUMBER, VARCHAR2, DATE), composite types (e.g., records, collections), and reference types.

□ **Cursors**: Cursors are pointers that allow for the retrieval of multiple rows from a SQL query. PL/SQL supports two types of cursors:

- **Implicit Cursors**: Automatically created byOracle when a SQL statement is executed.
- **Explicit Cursors**: Defined by the programmer for complex queries requiring row-by- row processing.

□ **Control Structures**: PL/SQL includes control structures such as loops (FOR, WHILE), conditional statements (IF, CASE), and exception handling constructs (EXCEPTION block).

Input :

The input will be a set of criteria to filter the data from the employees table. For example:

- Department ID
- Job Title

Output :

The output will be a list of employees that meet the specified criteria, including fields such as:

- Employee ID
- Name
- Job Title
- Salary

□ **Implementation**

Syntax :

Create the Employees Table

```
CREATE TABLE employees (
  employee_id NUMBER PRIMARY
  KEY, first_name VARCHAR2(50),
  last_name VARCHAR2(50),
  job_title
  VARCHAR2(50),
  department_id
  NUMBER,salary
  NUMBER
);
INSERT INTO employees VALUES (1, 'John', 'Doe', 'Developer', 10,
60000);INSERT INTO employees VALUES (2, 'Jane', 'Smith',
'Manager', 10, 80000);
INSERT INTO employees VALUES (3, 'Alice', 'Johnson', 'Developer', 20, 65000);
```

```
DECLARE

  v_department_id NUMBER := 10; -- Example input: set department

  ID to filter CURSOR emp_cursor IS

    SELECT employee_id, first_name, last_name, job_title,

    salary FROM employees

    WHERE department_id =

  v_department_id; v_employee_id

  employees.employee_id%TYPE;

  v_first_name

  employees.first_name%TYPE;

  v_last_name

  employees.last_name%TYPE; v_job_title

  employees.job_title%TYPE;
```

```
BEGIN

  OPEN emp_cursor;

  LOOP

    FETCH emp_cursor INTO v_employee_id, v_first_name, v_last_name, v_job_title,
v_salary;

    EXIT WHEN emp_cursor%NOTFOUND;

    DBMS_OUTPUT.PUT_LINE('ID: ' || v_employee_id || ', Name: ' || v_first_name
|| ' ' || v_last_name ||
```

- **Conclusion**

The PL/SQL block successfully retrieves and displays employee details based on the specified department ID. The use of cursors allows for the handling of multiple rows returned by the SQL query.

PRACTICAL NO. 10

Write and Execute SQL Queries for Triggers and Cursors

□ **Aim :**

To demonstratethe use oftriggers and cursors in SQL to automate actions and processmultiple rows of data in an Oracle database.

□ **Theory :**

1. Triggers

A trigger is a set of SQL statements that automatically execute (or "fire") in response to certain events on a particular table or view. Triggers can be used for various purposes, such as:

- Enforcing business rules.
- Maintaining audit trails.
- Automatically updating or validating data.

Types of Triggers:

- **BEFORE Trigger**: Executes before an insert, update, or delete operation.
- **AFTER Trigger**: Executes after the operation.
- **INSTEAD OF Trigger**: Executes in place of the operation, commonly used with views.

2. Cursors

A cursor is a database object used to retrieve a set of rows from a result set one at a time. Cursors can be particularly useful for processing multiple rows returned by a SQL query.

Types of Cursors:

- **Implicit Cursors**: Automatically created by Oracle for single SQL statements.
- **Explicit Cursors**: Defined by the user for complex queries that require row-by-row processing.

□ Implementation

Syntax : Create the Employees Table

```
CREATE TABLE employees (
  employee_id NUMBER PRIMARY
  KEY, first_name VARCHAR2(50),
  last_name VARCHAR2(50),
  department_id NUMBER,
  salary NUMBER
```

Syntax : Create a Trigger

```
CREATE TABLE salary_changes (
  change_id NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
  employee_id NUMBER,
  old_salary NUMBER,
  new_salary NUMBER,
  change_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
CREATE OR REPLACE TRIGGER
trg_salary_changeAFTER UPDATE OF salary ON
employees
FOR EACH ROW
BEGIN
  INSERT INTO salary_changes (employee_id, old_salary, new_salary)
  VALUES (:OLD.employee_id, :OLD.salary, :NEW.salary);
END;
```

Syntax : Using a Cursor

```
DECLARE
  v_salary_threshold NUMBER := 50000; -- Example input: salary threshold CURSOR
  emp_cursor IS
    SELECT employee_id, first_name, last_name, salary FROM
    employees
    WHERE salary > v_salary_threshold; v_employee_id
  employees.employee_id%TYPE;v_first_name
  employees.first_name%TYPE; v_last_name
  employees.last_name%TYPE; v_salary
  employees.salary%TYPE;
BEGIN
  OPEN emp_cursor;
  LOOP
    FETCH emp_cursor INTO v_employee_id, v_first_name, v_last_name, v_salary; EXIT
    WHEN emp_cursor%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE('ID: ' || v_employee_id || ', Name: ' || v_first_name || ' '
|| v_last_name ||
                ', Salary: ' || v_salary);
  END LOOP;
  CLOSE emp_cursor;
END;
```

- **Input**
  - **Trigger**: The trigger will be activated whenever an employee's salary is updated.
  - **Cursor**: The cursor will filter employees based on a salary threshold (in this example, 50,000).
- **Output**
  - **Trigger**: When an employee’s salary is updated, an entry is added to the salary_changes table, logging the old and new salary.
  - **Cursor**: The cursor retrieves and displays employee details for all employees whose salary is greater than 50,000.

- **Conclusion**

- **Triggers**: The trigger successfully logs changes to the salary field, providing a historical record of salary adjustments.
- **Cursors**: The cursor effectively retrieves and displays records based on the specified salary threshold.

PRACTICAL NO. 11

Design and Implementation of Database Systems or Packages for Applications Such as Office Automation, Hotel Management, HospitalManagement

□ **Aim :**

To design and implement a database system for a hotel management application that supports managing reservations, customers, rooms, and billing.

□ **Theory :**

1. Database Design

The hotel management system requires several key entities, which can be represented in the following tables:

- **Customers**: Stores information about hotel guests.
- **Rooms**: Contains details about each room in the hotel.
- **Reservations**: Tracks reservations made by customers.
- **Payments**: Records payment transactions for reservations.

2. Relationships

- A customer can have multiple reservations.
- A reservation is linked to one specific room.
- Payments are associated with specific reservations.

Database Schema

Syntax : Customers Table

```
CREATE TABLE customers ( customer_id NUMBER
   PRIMARY KEY,first_name VARCHAR2(50), last_name
   VARCHAR2(50),
   email VARCHAR2(100),phone
   VARCHAR2(15)
);
```

Syntax : Rooms Table

```
  CREATE TABLE rooms (

   room_id NUMBER PRIMARY KEY,

   room_type VARCHAR2(50),price NUMBER,

   status VARCHAR2(20) -- e.g., available, booked

);
```

Syntax : Reservations Table

```
CREATE TABLE reservations (

    reservation_id NUMBER PRIMARY KEY,

    customer_id NUMBER,

    room_id NUMBER,

    check_in DATE,

    check_out DATE,

    FOREIGN KEY (customer_id) REFERENCES
customers(customer_id),

    FOREIGN KEY (room_id) REFERENCES rooms(room_id)

);
```

Syntax : Payments Table

```
CREATE TABLE payments ( payment_id NUMBER PRIMARY KEY,

    reservation_id NUMBER,

    amount NUMBER, payment_date DATE,

    FOREIGN KEY (reservation_id)REFERENCES
reservations(reservation_id)

);
```

Implementation

Step 1: Create Tables

Run the SQL commands above to create the tables in your Oracle database.

Step 2: Create a Trigger

Let’s create a trigger to automatically update the room status to "booked" when a new reservation is made.

```
CREATE OR REPLACE TRIGGER
trg_update_room_status

AFTER INSERT ON  reservations

FOR EACH ROW

BEGIN

   UPDATE rooms

   SET status = 'booked'

   WHERE room_id = :NEW.room_id;

END;
```

Step 3: Using a Cursor

Create a PL/SQL block to retrieve and display all reservations along with customer names and room details.

```
DECLARE

  CURSOR res_cursor IS

     SELECT r.reservation_id, c.first_name, c.last_name, rm.room_type, r.check_in, r.check_out

     FROM reservations r

     JOIN customers c ON r.customer_id =

     c.customer_idJOIN rooms rm ON r.room_id =

     rm.room_id;

  v_reservation_id

  reservations.reservation_id%TYPE;v_first_name

  customers.first_name%TYPE; v_last_name

  customers.last_name%TYPE; v_room_type

  rooms.room_type%TYPE; v_check_in
```

```
BEGIN
   OPEN res_cursor;
   LOOP
      FETCH res_cursor INTO v_reservation_id, v_first_name, v_last_name,v_room_type,
v_check_in, v_check_out;
      EXIT WHEN res_cursor%NOTFOUND;
      DBMS_OUTPUT.PUT_LINE('Reservation ID: ' || v_reservation_id ||',
                     Name: ' || v_first_name || ' ' || v_last_name ||
                     ', Room Type: ' || v_room_type',
                     Check-in: ' || v_check_in || ', Check-
                     out: ' || v_check_out);
   END LOOP;
   CLOSE res_cursor;END
```

□ **Input**

1. **Trigger**: The trigger activates when a new reservation is inserted.
2. **Cursor**: The cursor retrieves all reservations with customer and room details.

□ **Output**

1. **Trigger**: When a reservation is made, the corresponding room's status updates to "booked."
2. **Cursor**: The cursor outputs details of all reservations, including customer names and room types.

□ **Conclusion**

- **Triggers**: The trigger effectively manages room status changes automatically with new reservations.
- **Cursors**: The cursor allows for the retrieval and display of comprehensive reservation details.

PRACTICAL NO. 12

Deployment of Forms, Reports Normalization, Query Processing Algorithms in the above Application Project

□ Aim :

To deploy forms and reports for the Hotel Management System, normalize the database schema, and outline query processing algorithms to optimize data retrieval and management.

□ Theory :

1. Deployment of Forms

Forms are user interfaces that allow users to input and manipulate data. In a hotel management system, you would typically have forms for:

- **Customer Registration**
- **RoomManagement**
- **Reservation Management**
- **Payment Processing**

Example: Customer Registration

This formwould include fields for:

- First Name
- Last Name
- Email
- Phone

This form can be implemented using web technologies (HTML/CSS/JavaScript) or desktop applications (Java Swing, .NET).

2. Reports

Reports are structured presentations of data, often for analysis and decision-making. In our hotel management system, we could have reports such as:

- Daily Reservations Report
- Room Occupancy Report
- Revenue Report

Reports can be generated using tools like Oracle Reports, Jasper Reports, or integrated within the application.

3. Normalization

Normalization is the process of organizing data to reduce redundancy and improve data integrity. The normalization process involves several normal forms:

- **First Normal Form (1NF)**: Ensures that each table column contains atomic values, and each record is unique.
- **Second Normal Form (2NF)**: Removes partial dependencies; all non-key attributes must depend on the entire primary key.
- **Third Normal Form (3NF)**: Removes transitive dependencies; non-key attributes should not depend on other non-key attributes.

Normalized Tables

- **Customers** (already in 3NF)
- **Rooms** (already in 3NF)
- **Reservations** (already in 3NF)
- **Payments** (already in 3NF)

Since the initial design was simple, it may already conform to 3NF. However, further analysis can be done to ensure no transitive dependencies exist.

4. Query Processing Algorithms

Query processing algorithms optimize the execution of SQL queries. Here are some key concepts and algorithms:

- **Query Parsing**: Analyzes the SQL query syntax and generates a query tree.
- **Query Optimization**: Rewrites the query in a more efficient way, possibly choosing different join algorithms (e.g., nested loops, hash joins).
- **Execution**: Executes the optimized query plan against the database.

Example Algorithms

- **Join Algorithms**:
    - **Nested Loop Join**: Useful for small datasets.
    - **Merge Join**: Efficient when both tables are sorted.
    - **Hash Join**: Suitable for larger datasets where indexing may not be feasible.

□ Implementation

Step 1: Forms

Assuming a web-based implementation, a simple HTML form for customer registration might look like this:

```
<form action="/registerCustomer" method="POST">
  <label for="firstName">First Name:</label>
  <input type="text" id="firstName" name="firstName" required>

  <label for="lastName">Last Name:</label>
  <input type="text" id="lastName" name="lastName" required>

  <label for="email">Email:</label>
  <input type="email" id="email" name="email" required>

  <label for="phone">Phone:</label>
  <input type="text" id="phone" name="phone" required>

  <button type="submit">Register</button>
</form>
```

Step 2: Reports

Using SQL, a sample query for a daily revenue report might look like this:

```
SELECT SUM(p.amount) AS
total_revenue, r.check_in

FROM payments p

JOINreservations r ON p.reservation_id
= r.reservation_id

GROUP BYr.check_in;
```

This SQL query aggregates payments based on the check-in date, providing a daily revenue total.

Step 3: Normalization

After analyzing the tables, you confirm they are already in 3NF. For example:

In the **Reservations** table, all attributes depend only on the reservation ID.

The **Payments** table ensures payments are linked to their respective reservations without redundant data.

Step 4: Query Processing Algorithms

In a practical implementation, the database management system (DBMS) automatically handles query parsingand optimization. However, as a developer, you can write efficient SQL queries and use proper indexing on frequently queried columns (like customer_id in reservations).

Input

1. **Forms**: Users can input customer data through the registration form.
2. **Reports**: SQL queries can be executed to generate various reports.
3. **Normalization**: Assessing and restructuring data for optimal integrity.
4. **Query Processing**: Use efficient algorithms to ensure quick data retrieval.

Output

1. **Forms**: A successful registration redirects users to a confirmation page.
2. **Reports**: SQL queryoutputs can be formatted as tables or charts showing daily revenue, occupancy rates, etc.
3. **Normalization**: All tables are optimized, ensuring no data anomalies.
4. **Query Processing**: Faster data retrieval due to optimized queries and indexing.

Conclusion

- **Forms**: Enable user-friendly data input.
- **Reports**: Provide valuable insights for management.
- **Normalization**: Enhances data integrity and reduces redundancy.
- **Query Processing**: Efficient algorithms and indexing lead to improved performance.

}

PRATICAL NO . 13

Distributed data base Management, creating web-page interfaces for database applications using servlet.

This tutorial assumes you have understanding on how JDBC application works. Before starting with database access through a servlet, make sure you have proper JDBC environment setup along with a database.In most cases, Dynamic web applications access a database to provide the client requested data. We can use Java standard database connection – JDBC in Servlets to perform database operations.

Servlet – Database connection

A Servlet can generate dynamic HTML by retrieving data from the database and sending it back to the client as a response. We can also update the database based on data passed in the client HTTP request. We will create a simple servlet to fetch/retrieve data from the database based on the client's request. In this example, we will be using Eclipse IDE and PostgreSQL database.

Import JDBC packages:

To access and process the Database operations, we need to import all the required "*java.sql*" packages in the program.

```
 import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
```

Register the JDBC Driver:

After importing all the packages, we need to register the JDBC driver which we installed into our program. Registering the driver tells the JVM to load the driver's class file into the memory so that we can implement the JDBC operations. We can register the driver by using "*Class.forName()*" method:

Class.forName():

```
try {
        // Register PostgreSQL Driver
        Class.forName("org.postgresql.Driver");
}
catch (ClassNotFoundException e) {
        System.out.println("Unable to load Driver class");
        // e.printStackTrace(); OR you
        // can directly print the stack trace
        System.exit(1);
}
```

Establish the Connection to Database:

Now, we need to establish the connection to the database using the "*DriverManager.getConnection()*" method.

```
String URL = "jdbc:postgresql://localhost/postgres";
String USER = "username";
String PASSWORD = "password";
Connection conn = DriverManager.getConnection(URL, USER, PASSWORD);
```

Create a JDBC Statement object:

After successful connection, prepare JDBC statement object using the Connection object.

```
Statement stmt = conn.createStatement();
```

Execute the SQL query:

Construct the SQL query based on the client request and execute the query using statement object.

```
stmt.executeQuery(sql);
```

Close Database connection:

After processing the required operations, finally, close all the database connection objects.

```
stmt.close();
conn.close();
```

Servlet – Packages

Servlets are the Java programs that run on the Java-enabled web server or application server. They are used to handle the request obtained from the webserver, process the request, produce the response, then send a response back to the webserver.

A package in servlets contains numerous classes and interfaces

Remember: *In order to create servlet in packages, use command "*

Types of Packages

There are two types of packages in Java Servlet that are providing various functioning features to servlet Applications. The two packages are as follows:

1. javax.servlet package
2. javax.servlet.http package

Type 1: javax.servlet package: This package of Servlet contains many servlet interfaces and classes which are capacity of handling any types of protocol sAnd This javax.servlet package containing large interfaces and classes that are invoked by the servlet or web server container as they are not specified with any protocol.

Type 2: javax.servlet.http package: This package of servlet contains more interfaces and classes which are capable of handling any specified http types of protocols on the servlet. This javax.servlet.http package containing many interfaces and classes that are used for http requests only for servlet

Interfaces and Classes in javax.servlet package