Empowerment Through Quality Technical Education
AJEENKYA DY PATIL SCHOOL OF ENGINEERING
Dr. D. Y. Patil Knowledge City, Charholi (Bk), Lohegaon, Pune – 412 105
Website: https://dypsoe.in/

LAB MANUAL

SOFTWARE LABORATORY-I

DATABASE MANAGEMENT SYSTEM (317523)

TE (AI&DS) 2020 COURSE

Course Coordinator

Prof. Poonam S. Nagale

DEPARTMENT OF

ARTIFICIAL INTELLIGENCE & DATA SCIENCE

Department of Artificial Intelligence & Data Science

Vision:

Imparting quality education in the field of Artificial Intelligence and Data Science

Mission:

- To include the culture of R and D to meet the future challenges in AI and DS.
- To develop technical skills among students for building intelligent systems to solve problems.
- To develop entrepreneurship skills in various areas among the students.
- To include moral, social and ethical values to make students best citizens of country.

Program Educational Outcomes:

1. To prepare globally competent graduates having strong fundamentals, domain knowledge, updated with modern technology to provide the effective solutions for engineering problems.
2. To prepare the graduates to work as a committed professional with strong professional ethics and values, sense of responsibilities, understanding of legal, safety, health, societal, cultural and environmental issues.
3. To prepare committed and motivated graduates with research attitude, lifelong learning, investigative approach, and multidisciplinary thinking.
4. To prepare the graduates with strong managerial and communication skills to work effectively as individuals as well as in teams.

Program Specific Outcomes:

1. Professional Skills- The ability to understand, analyze and develop computer programs in the areas related to algorithms, system software, multimedia, web design, networking, artificial intelligence and data science for efficient design of computer-based systems of varying complexities.

2. Problem-Solving Skills- The ability to apply standard practices and strategies in software project development using open-ended programming environments to deliver a quality product for business success.

3. Successful Career and Entrepreneurship- The ability to employ modern computer languages, environments and platforms in creating innovative career paths to be an entrepreneur and to have a zest for higher studies.

Table of Contents

Contents

1. Guidelines to manual usage..... 4
2. Laboratory Objective..... 8
3. Laboratory Equipment/Software..... 8
4. Laboratory Experiment list..... 9
4.1. Experiment No. 1..... 10
4.2. Experiment No. 2..... 12
4.3. Experiment No. 3..... 14
4.4. Experiment No. 4..... 16
4.5. Experiment No. 5..... 18
4.6. Experiment No. 6..... 20
5. Appendix..... 22

1. Guidelines to manual usage

This manual assumes that the facilitators are aware of collaborative learning methodologies.

This manual will provide a tool to facilitate the session on Digital Communication modules in collaborative learning environment.

The facilitator is expected to refer this manual before the session.

Icon of Graduate Attributes

K Applying Knowledge	**A** Problem Analysis	**D** Design & Development	**I** Investigation of problems
M Modern Tool Usage	**E** Engineer & Society	**E** Environment Sustainability	**T** Ethics
T Individual & Team work	**O** Communication	**M** Project Management & Finance	**I** Life-Long Learning

Disk Approach- Digital Blooms Taxonomy

1: Remembering / Knowledge

2: Comprehension / Understanding

3: Applying

4: Analyzing

5: Evaluating

6: Creating / Design

DATABASE MANAGEMENT SYSTEM (317523)

Program Outcomes:

1. **Engineering knowledge:** An ability to apply knowledge of mathematics, including discrete mathematics, statistics, science, computer science and engineering fundamentals to model the software application.
2. **Problem analysis:** An ability to design and conduct an experiment as well as interpret data, analyze complex algorithms, to produce meaningful conclusions and recommendations.
3. **Design/development of solutions**: An ability to design and development of software system, component, or process to meet desired needs, within realistic constraints such as economic, environmental, social, political, health & safety, manufacturability, and sustainability.
4. **Conduct investigations of complex problems**:An ability to use research based knowledge including analysis, design and development of algorithms for the solution of complex problems interpretation of data and synthesis of information to provide valid conclusion.
5. **Modern tool usage:** An ability to adapt current technologies and use modern IT tools, to design, formulate, implement and evaluate computer based system, process, by considering the computing needs, limits and constraints.
6. **The engineer and society:** An ability of reasoning about the contextual knowledge of the societal, health, safety, legal and cultural issues, consequent responsibilities relevant to IT practices.
7. **Environment and sustainability:** An ability to understand the impact of engineering solutions in a societal context and demonstrate knowledge of and the need for sustainable development.
8. **Ethics:** An ability to understand and commit to professional ethics and responsibilities and norms of IT practice.
9. **Individual and team work :**An ability to apply managerial skills by working effectively as an individual, as a member of a team, or as a leader of a team in multidisciplinary projects.
10. **Communication:** An ability to communicate effectively technical information in speech, presentation, and in written form
11. **Project management and finance:** An ability to apply the knowledge of Information Technology and management principles and techniques to estimate time and resources needed to complete engineering project.
12. **Life-long learning:** An ability to recognize the need for, and have the ability to engage in independent and life-long learning.

Course Name: Software Laboratory I- Database Management System

Course Code: (317523)

Course Outcomes

1. CO1:Implement SQL queries for given requirements, using different SQL concepts
2. CO2:Implement NoSQL queries using MongoDB
3. CO3:Design and develop application using database considering specific requirements

CO to PO Mapping:

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO1	-	1	2	2	-	-	3	2	1	-	-	1
CO2	-	1	2	2	-	2	3	1	-	1	-	1
CO3	2	2	3	3	1	-	3	-	2	1	2	2

CO to PSO Mapping:

	PSO1	PSO2	PSO3
CO1	2	1	1
CO2	2	2	1
CO3	2	2	1

2.Course Objectives:

- To Formalize and implement constraints in search problems
- To develop basic Database manipulation skills
- To develop skills to handle NoSQL database
- To learn understand to develop application using SQL or NoSQL databases.

3.**COURSE OUTCOMES**:

On completion of the course, learner will be able to–

CO1: Implement SQL queries for given requirements, using different SQL concepts

CO2: Implement NoSQL queries using MongoDB

CO3: Design and develop application using database considering specific requirements

2. Laboratory Equipment/Software

Operating System recommended: 64-bit Open source Linux or its derivative

Programming tools recommended: MYSQL/Oracle, MongoDB, ERD plus, ER Win and use suitable programming language/Tool for implementation

3. Laboratory Experiment list

Sr. No	Title
	Prerequisite practical assignments or installation (if any)
1	Fundamentals of Data Structures (210242), Data Structures and Algorithms (210253)
2	
	List of Assignments
1	SQL Queries: ● Design and Develop SQL DDL statements which demonstrate the use of SQL objects such as Table, View, Index, Sequence, Synonym, different constraints etc. Write at least 10 SQL queries on the suitable database application using SQL DML statements.
2	SQL Queries – all types of Join, Sub-Query and View: Write at least10 SQL queries for suitable database application using SQL DML statements. Note: Instructor will design the queries which demonstrate the use of concepts like all types of Join ,Sub-Query and View
3	MongoDB Queries: Design and Develop MongoDB Queries using CRUD operations. (Use CRUD operations, SAVE method.
4	Unnamed PL/SQLcode block: Use of Control structure and Exception handling is mandatory. Suggested Problem statement: Consider Tables: 1. Borrower (Roll_no, Name, Date_of_Issue, Name_of_Book, Status) 2. Fine (Roll_no, Date, Amt) ● Accept Roll_no and Name_of_Book from user. ● Check the number of days (from Date_of_Issue). ● If days are between 15 to 30 then fine amount will be Rs 5per day. ● If no. of days>30, per day fine will be Rs 50 per day and for days less than 30, Rs. 5 per day. ● After submitting the book, status will change from I to R. ● If condition of fine is true, then details will be stored into fine table. ● Also handles the exception by named exception handler or user define exception handler. OR ● MongoDB – Aggregation and Indexing: Design and Develop MongoDB Queries using aggregation and indexing with suitable example using MongoDB. MongoDB – Map-reduce operations: Implement Map-reduce operation with suitable example using MongoDB.
5	Cursors: (All types: Implicit, Explicit, Cursor FOR Loop, Parameterized Cursor) Write a PL/SQL block of code using parameterized Cursor that will merge the data available in the newly created table N_Roll_Call with the data available in the

	table O_Roll_Call. If the data in the first table already exists in the second table then that data should be skipped.
6	Database Connectivity: Write a program to implement MySQL/Oracle database connectivity with any front end language to implement Database navigation operations (add, delete, edit etc.)
	Group B.
7	Develop an application with following details: 1. Follow the same problem statement decided in Assignment-1 of Group A. 2. Follow the Software Development Life cycle and other concepts learnt in Software Engineering Course throughout the implementation. 3. Develop application considering: ● Front End: Python/Java/PHP/Perl/Ruby/.NET/ or any other language ● Backend : MongoDB/ MySQL/ Oracle / or any standard SQL / NoSQL database 4. Test and validate application using Manual/Automation testing. 5. Student should develop application in group of 2-3 students and submit the Project Report which will consist of documentation related to different phases of Software Development Life Cycle: ● Title of the Project, Abstract, Introduction ● Software Requirement Specification (SRS) ● Conceptual Design using ER features, Relational Model in appropriate Normalize form ● Graphical User Interface, Source Code ● Testing document ● Conclusion.
	Content Beyond Syllabus
1	
2	

Experiment No. 1

Aim:

SQL queries using Insert, Select, Update, delete with operators, functions, and set operator etc.

Objective:

To Write SQL queries using Insert, Select, Update, delete with operators, functions, and set operator etc. Use of SQL objects such as Table,View, Index, Sequence, Synonym, different constraints etc.

Theory:

Basic SQL Queries

SQL (Structured Query Language) is a standardized language used to manage and manipulate relational databases. There are several types of SQL queries that allow you to retrieve, insert, update, and delete data from a database. Here are the different types of SQL queries:

CREATE YOUR DATABASE:

```
CREATE DATABASE your_database_name;
```

Replace **your_database_name** with the desired name for your database.

CREATE a table within DATABASE:

USE your_database_name;

```
CREATE TABLE your_table_name (
column1 datatype1 constraints,
column2 datatype2 constraints,
...
);
```

SELECT Query:

The **SELECT** query is used to retrieve data from one or more database tables. It is the most commonly used query in SQL. With the SELECT query, you can specify the columns you want to retrieve and apply conditions using the
WHERE clause to filter the data. Here's an example:

```
SELECT column1, column2
FROM table
WHERE condition;
```

· **column1, column2**, ...: The columns you want to retrieve data from. You can use an asterisk (*) to select all columns in the table.

· **table:** The name of the table from which you want to retrieve data.

· **WHERE** condition: An optional clause that allows you to filter the data based on specific conditions. Only rows that satisfy the condition will be included in the result.

INSERT Query:

The INSERT query is used to insert new records into a database table. It allows you to specify the values for each column or use a SELECT statement to insert data from another table. Here's an example:

```
INSERT INTO table (column1, column2)
VALUES (value1, value2);
```

- **table**: The name of the table into which you want to insert data.
- **column1, column2**, ...: The columns into which you want to insert data.
- **value1, value2**, ...: The values you want to insert into the corresponding columns
-

UPDATE Query:

The UPDATE query is used to modify existing records in a database table. It allows you to update one or more columns with new values based on a specified condition. Here's an example:

UPDATE table:

SET column1 = value1, column2 = value2 WHERE condition;

- **table:** The name of the table you want to update.
- **SET** column1 = value1, column2 = value2, ...: Specifies the columns you want to update and the new values you want to assign to them.
- **WHERE** condition: An optional clause that specifies which rows you want to update. If omitted, all rows in the table will be updated

DELETE Query:

The DELETE query is used to remove records from a database table. It allows you to delete one or more rows based on a specified condition. Here's an example:

```
DELETE FROM
table WHERE
condition;
```

table: The name of the table from which you want to delete data.
WHERE condition: An optional clause that specifies which rows you want to delete. If omitted, all rows in the table will be deleted

ORDER BY:

The ORDER BY query is used to sort the result of a SELECT query in ascending or descending order.

```
SELECT column1, column2, ...
FROM table_name
```

```
ORDER BY column1 ASC/DESC, column2 ASC/DESC, ...;
```

- **column1, column2, ...**: The columns by which you want to sort the result.
- **ASC**: Ascending order (default).
- **DESC**: Descending order.

GROUP BY:

The GROUP BY query is used to group rows based on the values in one or more columns.

```
SELECT column1, column2, ... FROM
table_name
GROUP BY column1, column2, ...;
```

column1, column2, ...: The columns by which you want to group the result.

CONSTRAINTS:

In MySQL 8.0, you can create various constraints to enforce data integrity rules on your database tables. Constraints ensure that data in the tables adheres to certain rules and restrictions, preventing the insertion of incorrect or invalid data. Here are examples of some commonly used constraints in MySQL 8.0:

Primary Key Constraint: The primary key constraint ensures that a column (or a combination of columns) uniquely identifies each row in the table. It also automatically creates an index on the primary key column for faster data retrieval.

```
CREATE TABLE employees ( employee_id INT
   PRIMARY KEY, first_name VARCHAR(50),
   last_name VARCHAR(50), department
   VARCHAR(50),
   salary DECIMAL(10, 2), hire_date DATE
);
```

Foreign Key Constraint: The foreign key constraint establishes a relationship between two tables. It ensures that the values in a specific column (or combination of columns) in one table match the values of the primary key column in another table.

```
CREATE TABLE departments ( department_id INT PRIMARY KEY, department_name VARCHAR(50)
);

CREATE TABLE employees ( employee_id INT PRIMARY KEY, first_name VARCHAR(50), last_name
VARCHAR(50), department_id INT,
salary DECIMAL(10, 2), hire_date DATE,
FOREIGN KEY (department_id) REFERENCES departments(department_id)
);
```

Unique Constraint: The unique constraint ensures that the values in the specified column (or combination of columns) are unique across the table.

```
CREATE TABLE students ( student_id INT PRIMARY KEY, student_name VARCHAR(50), email
VARCHAR(100) UNIQUE,
age INT
);
```

Check Constraint: The check constraint enforces a condition on the data being inserted into the table, ensuring that the condition evaluates to true.

```
CREATE TABLE products ( product_id INT PRIMARY KEY, product_name VARCHAR(100), price
DECIMAL(10, 2),
quantity INT,
CHECK (price >= 0 AND quantity >= 0)
);
```

Not Null Constraint: The not null constraint ensures that a column cannot contain NULL values.

```
CREATE TABLE orders ( order_id INT PRIMARY KEY, order_date DATE NOT NULL, customer_id INT
NOT NULL,
total_amount DECIMAL(10, 2) NOT NULL
);
```

SEQUENCE

In MySQL, the concept of sequences, is not supported. MySQL does not have a dedicated "SEQUENCE" object to create sequences for generating unique numeric values automatically.

However, MySQL provides an alternative way to achieve similar functionality using the AUTO_INCREMENT attribute for a column in a table. When you define a column with the AUTO_INCREMENT attribute, MySQL automatically assigns a unique value to that column whenever a new row is inserted into the table.

Example to create a sequence in MYSQL:

Create a table with an auto-incrementing column:

```
CREATE TABLE your_table_name (
column_name INT AUTO_INCREMENT PRIMARY KEY, other_column1 datatype1,
other_column2 datatype2,
...
);
```

Example :

```
CREATE TABLE employees (
employee_id INT AUTO_INCREMENT PRIMARY KEY, first_name VARCHAR(50),
last_name VARCHAR(50),

department VARCHAR(50), salary DECIMAL(10, 2), hire_date DATE
);
```

In this example, the employee_id column is marked as INT AUTO_INCREMENT PRIMARY KEY, making it an auto-incrementing primary key for the "employees" table.
Insert records into the table without specifying values for the auto-increment column:

```
INSERT INTO employees (first_name, last_name, department, salary, hire_date) VALUES ('John', 'Doe',
'HR', 55000, '2023-07-15');
```

In MySQL, an index is a database object that improves the retrieval speed of data from database tables. It works similarly to an index in a book, allowing the database management system (DBMS) to quickly find the location of data within a table. When you create an index on one or more columns, MySQL creates a data structure that organizes the values in those columns in a way that makes searching and filtering operations more efficient.
Indexes are particularly useful for large tables or when you frequently search, sort, or filter data based on certain columns. By using indexes, MySQL can avoid scanning the entire table and instead perform a more targeted search, significantly reducing query execution time.

Creating an Index: You can create an index on one or more columns using the CREATE INDEX statement:

CREATE INDEX index_name ON table_name (column1, column2, ...);

•index_name: A name for the index. Choose a meaningful name to represent the purpose of the index.
•table_name: The name of the table on which you want to create the index.
•(column1, column2, ...): The column(s) you want to include in the index. You can include multiple columns for a composite index.

SYNONYM

In MySQL, a synonym, is not supported. In MySQL, there is no specific "SYNONYM" keyword or object to create aliases for database objects.

However, you can achieve similar functionality to synonyms in MySQL by creating a VIEW. A view is a virtual table derived from the result of a SELECT query. It can be used to provide an alias or a more user-friendly name for a table, or it can be used to restrict access to certain columns of a table.
Let's see how you can use a VIEW as an alternative to synonyms in MySQL: CREATE VIEW view_name AS
SELECT column1, column2, ... FROM table_name
WHERE condition;

•view_name: The name of the view you want to create.
•column1, column2, ...: The columns you want to include in the view. You can specify any columns from the table or use aggregate functions to transform the data as needed.
•table_name: The name of the original table from which you want to create the view.
•WHERE condition: An optional clause that allows you to filter the data from the original table and include only specific rows in the view.

Use the VIEW:
Once the view is created, you can use it in your queries just like you would use a regular table: SELECT * FROM view_name;
The above query will retrieve data from the VIEW called view_name, which is essentially a representation of the data from the original table, filtered or transformed as specified in the CREATE VIEW statement.

Views are read-only, meaning you cannot perform INSERT, UPDATE, or DELETE operations directly on them. Any changes to the data should be made on the original tables. However, views can provide a convenient and secure way to simplify complex queries, restrict access to certain columns, or create aliases for tables, which is similar to what synonyms offer in other database systems.

Input:

Problem Statement: Account(Acc_no, branch_name,balance) branch(branch_name,branch_city,assets) customer(cust_name,cust_street,cust_city) Depositor(cust_name,acc_no) Loan(loan_no,branch_name,amount) Borrower(cust_name,loan_no)

Solve following query:

Create above tables with appropriate constraints like primary key, foreign key, check constrains, not null etc.
Q1. Find the names of all branches in loan relation.

Q2. Find all loan numbers for loans made at Akurdi Branch with loan amount > 12000.

Q3. Find all customers who have a loan from bank. Find their names, loan_no and loan amount. Q4. List all customers in alphabetical order who have loan from Akurdi branch.
Q5. Find all customers who have an account or loan or both at bank. Q6. Find all customers who have both account and loan at bank.
Q7. Find all customer who have account but no loan at the bank. Q8. Find average account balance at Akurdi branch.
Q9. Find the average account balance at each branch Q10. Find no. of depositors at each branch.
Q11. Find the branches where average account balance > 12000.
Q12. Find number of tuples in customer relation.
Q13. Calculate total loan amount given by bank.
Q14. Delete all loans with loan amount between 1300 and 1500.
Q15. Delete all tuples at every branch located in Nigdi.
Q.16. Create synonym for customer table as cust.
Q.17. Create sequence roll_seq and use in student table for roll_no column

OUTPUT: Various SQL queries like select, insert, update, delete have been successfully executed on table. Also use of SQL objects such as Table,View, Index, Sequence, Synonym, different constraints etc.

Conclusion:

Thus, we have successfully studied DDL,DML commands and implemented SQL Queries.

Outcome:

Ability to write DDL and DML commands and their uses.

QUESTIONS:

1. Classify of SQL commands in detail.
2. What is the difference between DDL & DML?
3. Explain the following: - 1. Order by
   2. Group By
   3. Having
   4. Where

4. Explain aggregate function in detail.
5. What is SQL? Write characteristics and Advantages of it?
6. Explain use of sequence and synonym queries.

Experiment No. 2

Aim:

SQL Queries - all types of Join, Sub-Query and View

Objective:

- To understand the use of SQL joins
- To understand the use of sub queries in SQL

Theory:

SQL JOIN

A JOIN clause is used to combine rows from two or more tables, based on a related column between them. Let's look at a selection from the "Orders" table:

OrderID	CustomerID	OrderDate
10308	2	1996-09-18
10309	37	1996-09-19
10310	77	1996-09-20

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

CustomerID	CustomerName	ContactName	Country
1	Alfreds Futterkiste	Maria Anders	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mexico

Note that the "CustomerID" column in the "Orders" table refers to the "CustomerID" in the "Customers" table. The relationship between the two tables above is the "CustomerID" column.Then, we can create the following SQL statement (that contains an INNER JOIN), that selects records that have matching values in both tables:

Example:

SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate FROM Orders INNER JOIN Customers ON Orders.CustomerID=Customers.CustomerID;

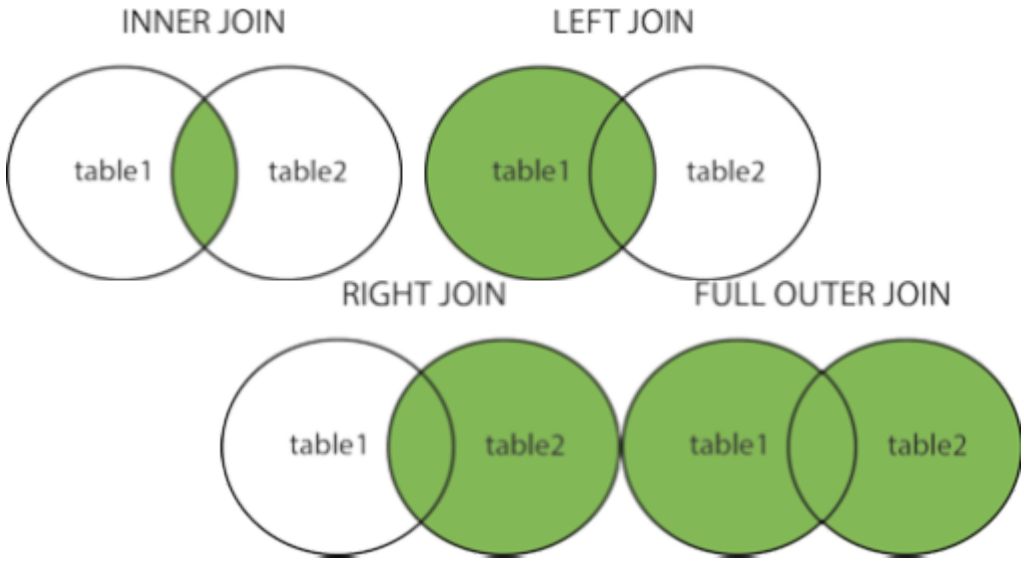
and it will produce something like this:

OrderID	CustomerName	OrderDate
10308	Ana Trujillo Emparedados y helados	9/18/1996
10365	Antonio Moreno Taquería	11/27/1996
10383	Around the Horn	12/16/1996
10355	Around the Horn	11/15/1996
10278	Berglunds snabbköp	8/12/1996

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



SQL Views: SQL CREATE VIEW Statement

In SQL, a view is a virtual table based on the result-set of an SQL statement. A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database. You can add SQL functions, WHERE, and JOIN statements to a view and present the data as if the data were coming from one single table.

CREATE VIEW Syntax:

CREATE VIEW view_name AS SELECT column1, column2, ... FROM table_name

WHERE condition;

SQL CREATE VIEW Examples:

The view "Product List" lists all active products (products that are not discontinued) from the "Products" table. The view is created with the following SQL:

CREATE VIEW ProductList AS SELECT ProductID, ProductName FROM Products

WHERE Discontinued = No;

Then, we can query the view as follows: SELECT * FROM ProductList;

SQL Updating a View:

You can update a view by using the following syntax:

SQL CREATE OR REPLACE VIEW Syntax CREATE OR REPLACE VIEW view_name AS SELECT column1, column2, ...

FROM table_name WHERE condition;

Now we want to add the "Category" column to the "Product List" view. We will update the view with the following SQL:

CREATE OR REPLACE VIEW Product List AS

SELECT ProductID, ProductName, Category FROM

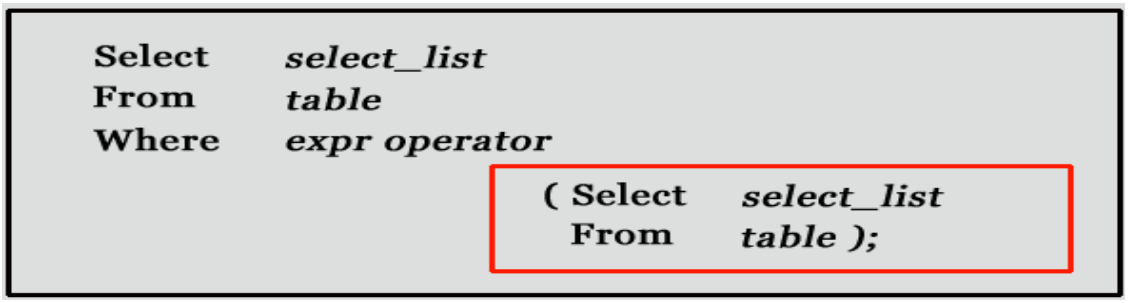
Products WHERE Discontinued = No

Subqueries:

A subquery is a query nested within another query. It allows you to use the result of one query as a condition or filter in another query.

- A subquery may occur in :
- - A SELECT clause
- - A FROM clause
- - A WHERE clause
- In MySQL subquery can be nested inside a SELECT, INSERT, UPDATE, DELETE, SET, or DO statement or inside another subquery.
- A subquery is usually added within the WHERE Clause of another SQL SELECT statement.
- You can use the comparison operators, such as >, <, or =. The comparison operator can also be a multiple-row operator, such as IN, ANY, SOME, or ALL.
- A subquery can be treated as an inner query, which is a SQL query placed as a part of another query called as outer query.
- The inner query executes first before its parent query so that the results of the inner query can be passed to the outer query

Subquery Syntax:



- The subquery (inner query) executes once before the main query (outer query) executes.
- The main query (outer query) use the subquery result.

Subquery syntax as specified by the SQL standard and supported in

MySQL DELETE FROM t1 WHERE s11 > ANY

(SELECT COUNT(*) /* no hint */ FROM t2

WHERE NOT EXISTS (SELECT * FROM t3 WHERE ROW(5*t2.s1,77)=

(SELECT 50,11*s1 FROM t4 UNION SELECT 50,77 FROM (SELECT * FROM t5) AS t5)));

A subquery can return a scalar (a single value), a single row, a single column, or a table (one or more rows of one or more columns). These are called scalar, column, row, and table subqueries.

Subqueries: Guidelines

There are some guidelines to consider when using subqueries :

- A subquery must be enclosed in parentheses.

- Use single-row operators with single-row subqueries, and use multiple-row operators with multiple-row subqueries.
- If a subquery (inner query) returns a null value to the outer query, the outer query will not return any rows when using certain comparison operators in a WHERE clause.

Types of Subqueries

- The Subquery as Scalar Operand
- Comparisons using Subqueries
- Subqueries with ALL, ANY, IN, or SOME
- Row Subqueries
- Subqueries with EXISTS or NOT EXISTS
- Correlated Subqueries
- Subqueries in the FROM Clause

STORED PROCEDURES:

SELECT COUNT(column) FROM table

WHERE condition;

Stored procedures are precompiled SQL statements that are stored in a database. They allow you to encapsulate a series of SQL statements into a single executable procedure. Stored procedures can accept parameters, perform complex calculations, and return results. They are commonly used to improve performance and maintainability. Here's an example:

CREATE PROCEDUREprocedure_name (parameter1, parameter2)

AS BEGIN -- SQL statements END;

These are some of the most commonly used SQL queries. By combining and using these queries effectively, you can manage and manipulate data in a relational database.

Applications:

1. To eliminate unimportant words
2. To allow applications to focus on the important words
3. To drop common words

Input:

Problem Statement:

1. Create following Tables customer_master

(cust_no, fname, lname)

address_details (cust_no, add1, add2, state, city, pincode)

Retrieve the address of customer whose Fname as 'xyz' and Lname as 'pqr'

2. Create following Tables customer_master

(cust_no, fname, lname)

acc_fixdeposite_cust_details (custno, acc_fd_no)

fixdeposite_dets(fd_sr_no, amt)

List the customer holding fixed deposit of amount more than 5000

2. Create following Tables

emp_mstr (e_mpno, f_name, l_name, m_name, dept, desg, branch_no)
branch_master (branch_name, branch_no)

List the employee details along with branch names to which they belong

3. Create following Tables

emp_master (emp_no, f_name, l_name, m_name, dept)
contact_details(emp_no, cntc_type, cntc_data)

List the employee details along with contact details using left outer join & right join

4. Create following Tables

customer_master (cust_no, fname, lname, branch_no)

address_details (cust_no, pincode)

branch(branch_no, branch name, pincode)

List the customer who do not have bank branches in their vicinity.

5. **a) Create View on borrower table by selecting any two columns and perform insert update delete operations**

b]Create view on borrower & depositor table by selecting any one column from each table, perform insert, update, delete operations

Output: We Learn how to execute the SQL queries using concepts like all types of Join, Sub-Query and View

Conclusion:

Various SQL queries including joins, views, aggregate functions for suitable database application using SQL DML statements are studied and executed.

Outcome:

Able to write SQL queries using types of Join,Sub-Query and view

Questions:

1. Explain Join Function.
2. Enlist the different types of join operations.
3. Explain natural Join with example.
4. Explain Outer join, explain with example.
5. What is the use of nested Query. Explain with Example.

Experiment No. 3

Aim:

MongoDB Queries using CRUD operations. (Use CRUDoperations, SAVE method, logical operators etc)

Objective: To Write the MongoDB Queries using CRUD operations. (Use CRUDoperations, SAVE method, logical operators etc)

Theory:

MongoDB is a cross-platform, document oriented database that provides, high performance, high availability, and easy scalability. MongoDB works on concept of collection and document.

Database

Database is a physical container for collections. Each database gets its own set of files on the file system. A single MongoDB server typically has multiple databases.

Collection

Collection is a group of MongoDB documents. It is the equivalent of an RDBMS table. A collection exists within a single database. Collections do not enforce a schema. Documents within a collection can have different fields. Typically, all documents in a collection are of similar or related purpose.

Document

A document is a set of key-value pairs. Documents have dynamic schema. Dynamic schema means that documents in the same collection do not need to have the same set of fields or structure, and common fields in a collection's documents may hold different types of data.

The following table shows the relationship of RDBMS terminology with MongoDB

RDBMS	MongoDB
Database	Database
Table	Collection
Tuple/Row	Document
column	Field
Table Join	Embedded Documents
Primary Key	Primary Key (Default key _id provided by mongodb itself)
Database Server and Client	
Mysqld/Oracle	mongod
mysql/sqlplus	mongo

CRUD is the basic operation of Mongodb ,it stands CREATE , READ , UPDATE, DELETE.

MongoDB ─ 1. Create Collection

The createCollection() Method

MongoDB db.createCollection(name, options) is used to create collection. Basic syntax of createCollection() command is as follows: db.createCollection(name, options)

In the command, name is name of collection to be created. Options are a document and are used to specify configuration of collection.

Parameter	Type	Description
Name	String	Name of the collection to be created
Options	Document	(Optional) Specify options about memory size and indexing

Options parameter is optional, so you need to specify only the name of the collection. Following is the list of options you can use:

Field	Type	Description
capped	Boolean	(Optional) If true, enables a capped collection. Capped collection is a fixed size collection that automatically overwrites its oldest entries when it reaches its maximum size. **If you specify true, you need to specify size parameter also.**
autoIndexID	Boolean	(Optional) If true, automatically create index on _id field. Default value is false.
size	number	(Optional) Specifies a maximum size in bytes for a capped collection. **If capped is true, then you need to specify this field also.**
max	number	(Optional) Specifies the maximum number of documents allowed in the capped collection.

While inserting the document, MongoDB first checks size field of capped collection, then it checks max field.

Examples

Basic syntax of createCollection() method without options is as follows:

```
>use test
switched to db test
>db.createCollection("mycollection")
 { "ok" : 1 }
>
```

You can check the created collection by using the command show collections.

```
>show collections mycollection
 system.indexes
```

2. READ-The find() Method

To query data from MongoDB collection, you need to use MongoDB's find()method. Syntax

The basic syntax of find() method is as follows:

```
>db.COLLECTION_NAME.find()
```

find()method will display all the documents in a non-structured way. The pretty() Method

To display the results in a formatted way, you can use pretty() method. Syntax

```
>db.mycol.find().pretty() Example
>db.mycol.find().pretty()
{
"_id": ObjectId(7df78ad8902c), "title": "MongoDB Overview",
"description": "MongoDB is no sql database", "by": "tutorials point",
"url": "http://www.tutorialspoint.com", "tags": ["mongodb", "database", "NoSQL"], "likes": "100"
}
>
```

Apart from find() method, there is findOne() method, that returns only one document.

3. UPDATE

MongoDB's update() and save() methods are used to update document into a collection.

The update() method updates the values in the existing document while the save() method replaces the existing document with the document passed in save() method.

MongoDB Update() Method

The update() method updates the values in the existing document. The basic syntax of update() method is as follows:

>db.COLLECTION_NAME.update(SELECTIOIN_CRITERIA, UPDATED_DATA)

Example

Consider the mycol collection has the following data.

{ "_id" : ObjectId(5983548781331adf45ec5), "title":"MongoDB Overview"}

{ "_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}

{ "_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point Overview"} Following example will set the new title 'New MongoDB Tutorial' of the documents whose title is 'MongoDB Overview'.

>db.mycol.update({'title':'MongoDB Overview'},{$set:{'title':'New MongoDB Tutorial'}})

>db.mycol.find()

{ "_id" : ObjectId(5983548781331adf45ec5), "title":"New MongoDB Tutorial"}

{ "_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}

{ "_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point Overview"}

>

By default, MongoDB will update only a single document. To update multiple documents, you need to set a parameter 'multi' to true.

>db.mycol.update({'title':'MongoDB Overview'},

{$set:{'title':'New MongoDB Tutorial'}},{multi:true})

MongoDB Save() Method

The save() method replaces the existing document with the new document passed in the save() method. The basic syntax of MongoDB save() method is −

>db.COLLECTION_NAME.save({_id:ObjectId(),NEW_DATA}) Example

Following example will replace the document with the _id '5983548781331adf45ec7'.

>db.mycol.save(

{

"_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point New Topic",

"by":"Tutorials Point"

})

>db.mycol.find()

{ "_id" : ObjectId(5983548781331adf45ec5), "title":"Tutorials Point New Topic", "by":"Tutorials Point"}

{ "_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}

{ "_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point Overview"}

4. DELETE-The remove() Method

MongoDB's remove() method is used to remove a document from the collection.

remove() method accepts two parameters. One is deletion criteria and second is justOne flag.

• deletion criteria: (Optional) deletion criteria according to documents will be removed.

• justOne: (Optional) if set to true or 1, then remove only one document. Basic syntax of remove() method is as follows:

>db.COLLECTION_NAME.remove(DELLETION_CRITTERIA)

Example

Consider the mycol collection has the following data.

{ "_id" : ObjectId(5983548781331adf45ec5), "title":"MongoDB Overview"}
{ "_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}

{ "_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point Overview"} Following example will remove all the documents whose title is 'MongoDB Overview'.

>db.mycol.remove({'title':'MongoDB Overview'})

>db.mycol.find()

{ "_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}

{ "_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point Overview"}

LOGICAL OPERATORS:

Syntax

AND in MongoDB

In the find() method, if you pass multiple keys by separating them by ',' then MongoDB treats it as AND condition. Following is the basic syntax of AND −

>db.mycol.find({key1:value1, key2:value2}).pretty() Example

Following example will show all the tutorials written by 'tutorials point' and whose title is 'MongoDB Overview'.

>db.mycol.find({"by":"tutorials point","title": "MongoDB Overview"}).pretty()

{

"_id": ObjectId(7df78ad8902c), "title": "MongoDB Overview",

"description": "MongoDB is no sql database", "by": "tutorials point",

"url": "http://www.tutorialspoint.com", "tags": ["mongodb", "database", "NoSQL"], "likes": "100"

}>

For the above given example, equivalent where clause will be ' where by='tutorials point' AND title = 'MongoDB Overview' '. You can pass any number of key, value pairs in find clause.

OR in MongoDB

Syntax : To query documents based on the OR condition, you need to use $or keyword. Following is the basic syntax of OR −

>db.mycol.find({ $or: [{key1: value1}, {key2:value2}] }).pretty()

Example will show all the tutorials written by 'tutorials point' or whose title is 'MongoDB Overview'.

>db.mycol.find({$or:[{"by":"tutorials point"},{"title": "MongoDB Overview"}]}).pretty()

{ "_id": ObjectId(7df78ad8902c), "title": "MongoDB Overview",

"description": "MongoDB is no sql database", "by": "tutorials point",

"url": "http://www.tutorialspoint.com", "tags": ["mongodb", "database", "NoSQL"], "likes": "100" }

Using AND and OR Together Example

The following example will show the documents that have likes greater than 100 and whose title is either 'MongoDB Overview' or by is 'tutorials point'. Equivalent SQL where clause is 'where likes>10 AND (by = 'tutorials point' OR title = 'MongoDB Overview')'

```
>db.mycol.find({"likes": {$gt:10}, $or: [{"by":"tutorials
    point"},{"title": "MongoDB Overview"}]}).pretty()
```

{ "_id": ObjectId(7df78ad8902c), "title": "MongoDB Overview",

"description": "MongoDB is no sql database", "by": "tutorials point",

"url": "http://www.tutorialspoint.com", "tags": ["mongodb", "database", "NoSQL"], "likes": "100" }

Applications:

Input:

Problem Statement:

1. Create Database DYPCOE
2. Create following Collections Teachers(Tname,dno,dname,experience,salary,date_of_joining) Students(Sname,roll_no,class)
3. Find the information about all teachers
4. Find the information about all teachers of computer department
5. Find the information about all teachers of computer,IT,and e&TC department
6. Find the information about all teachers of computer,IT,and E&TC department having salary greater than or equal to 10000/-
7. Find the student information having roll_no = 2 or Sname=xyz
8. Update the experience of teacher-praveen to 10years, if the entry is not available in database consider the entry as new entry.
9. Update the deparment of all the teachers working in IT department to COMP
10. find the teachers name and their experience from teachers collection
11. Using Save() method insert one entry in department collection
12. Using Save() method change the dept of teacher praveen to IT

13. Delete all the documents from teachers collection having IT dept.

display with pretty() method, the first 3 documents in teachers collection in ascending order

Output: Able to write MongoDB Queries using CRUD operations

Conclusion: Thus we have studied MongoDB Queries using CRUD operations.

Questions:

1.
2.
3.
4.
5.

Experiment No. 4

Aim: A PL/SQL block of code (Use of Control structure and Exception handling)

Objective: To write a PL/SQL block of code using Control structure and Exception handling.

Theory:

The PL/SQL architecture mainly consists of following 3 components:

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

Server PL/SQL block:

- This is the component which has the actual PL/SQL code.
- This consists of different sections to divide the code logically (declarative section for declaring purpose, execution section for processing statements, exception handling section for handling errors)
- It also contains the SQL instruction that used to interact with the database server.
- All the PL/SQL units are treated as PL/SQL blocks, and this is the starting stage of the architecture which serves as the primary input.
- Following are the different type of PL/SQL units.

o Anonymous Block

o Function

o Library

o Procedure

o Package Body

o Package Specification

o Trigger

o Type

o Type Body

PL/SQL Engine:

- PL/SQL engine is the component where the actual processing of the codes takes place.

- PL/SQL engine separates PL/SQL units and SQL part in the input (as shown in the image below).
- The separated PL/SQL units will be handled with the PL/SQL engine itself.
- The SQL part will be sent to database server where the actual interaction with database takes place.
- It can be installed in both database server and in the application server.

Database Server:

- This is the most important component of Pl/SQL unit which stores the data.
- The PL/SQL engine uses the SQL from PL/SQL units to interact with the database server.
- It consists of SQL executor which actually parses the input SQL statements and execute the same.

Advantage of Using PL/SQL

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

Features of PL/SQL:-

1) We can define and use variables and constants in PL/SQL.

2) PL/SQL provides control structures to control the flow of a program. The control structures supported by PL/SQL are if..Then, loop, for..loop and others.

3) We can do row by row processing of data in PL/SQL.PL/SQL supports row by row processing using the mechanism called cursor.

4) We can handle pre-defined and user-defined error situations. Errors are warnings and called as exceptions in PL/SQL.

5) We can write modular application by using sub programs.

The structure of PL/SQL program:-

The basic unit of code in any PL/SQL program is a block. All PL/SQL programs are composed of blocks. These blocks can be written sequentially.

The structure of PL/SQL block:-

DECLARE

Declaration section BEGIN

Executable section

EXCEPTION Exception

handling section END;

Where

1) Declaration section

PL/SQL variables, types, cursors, and local subprograms are defined here.

2) Executable section

Procedural and SQL statements are written here. This is the main section of the block. This section is required.

3) Exception handling section

Error handling code is written

here

This section is optional whether it is defined within body or outside body of program.

Conditional statements and Loops used in PL/SQL

Conditional statements check the validity of a condition and accordingly execute a set of statements. The conditional statements supported by Pl/SQL is

1) IF..THEN

2) IF..THEN..ELSE

3) IF..THEN..ELSIF

1.IF..THEN

Syntax1:- If condition THEN Statement list END IF;

2.IF..THEN..ELSE

Syntax 2:-

IF condition

THEN Statement

list

ELSE

Statements

1) **IF..THEN..ELSIF**

END IF;

Syntax 3:-

If condition THEN

Statement list ELSIF condition THEN

Statement list

ELSE

Statement list

END IF; END

IF;

2) **CASE Expression** :CASE expression can also be used to control the branching logic within PL/SQL blocks. The general syntax is

CASE

WHEN <expression> THEN <statements>; WHEN <expression> THEN <statements>;

. ELSE

<statements>; END CASE;

Here expression in WHEN clause is evaluated sequentially. When result of expression is TRUE, then corresponding set of statements are executed and program flow goes to END CASE.

ITERATIVE Constructs : Iterative constructs are used to execute a set of statements respectively. The iterative constructs supported by PL/SQL are follows:

1) SIMPLE LOOP

2) WHILE LOOP

3) FOR LOOP

1) **The Simple LOOP** : It is the simplest iterative construct and has syntax like:

LOOP Statements

END LOOP;

The LOOP does not facilitate a checking for a condition and so it is an endless loop. To end the iterations, the EXIT statement can be used.

LOOP

<statement list>

IF condition THEN EXIT;

END IF;

END LOOP;

The statements here is executable statements,which will be executed repeatedly until the condition given if IF..THEN evaluates TRUE.

2) THE WHILE LOOP

The WHILE…LOOP is a condition driven construct i.e the condition is a part of the loop construct and

not to be checked separately. The loop is executed as long as the condition evaluates to TRUE. The syntax is:-

WHILE condition LOOP

Statements END LOOP;

The condition is evaluated before each iteration of loop. If it evaluates to TRUE, sequence of statements are executed. If the condition is evaluated to FALSE or NULL, the loop is finished and the control resumes after the END LOOP statement.

3) **THE FOR LOOP** :The number of iterations for LOOP and WHILE LOOP is not known in advance. THE number of iterations depends on the loop condition. The FOR LOOP can be used to have a definite numbers of iterations.

The syntax is:-

Where

For loop counter IN [REVERSE] Low bound..High bound LOOP

Statements; End loop;

- loop counter –is the implicitly declared index variable as BINARY_INTEGER.
- Low bound and high bound specify the number of iteration .
- Statements:-Are the contents of the loop

EXCEPTIONS:- Exceptions are errors or warnings in a PL/SQL program.PL/SQL implements error handling using exceptions and exception handler.

Exceptions are the run time error that a PL/SQL program may encounter. There are two types of exceptions

1) Predefined exceptions

2) User defined exceptions

1) Predefined exceptions:- Predefined exceptions are the error condition that are defined by ORACLE. Predefined exceptions cannot be changed. Predefined exceptions correspond to common SQL errors. The predefined exceptions are raised automatically whenever a PL/SQL program violates an ORACLE rule.

2) User defined Exceptions:- A user defined exceptions is an error or a warning that is defined by the program. User defined exceptions can be define in the declaration section of PL/SQL block.

User defined exceptions are declared in the declarative section of a PL/SQL block. Exceptions have a type Exception and scope.

Syntax :

DECLARE

<Exception Name> EXCEPTION; BEGIN

….

RAISE <Exception Name>

…

EXCEPTION

WHEN <Exception name> THEN

<Action>

END;

Exception Handling

A PL/SQL block may contain statements that specify exception handling routines. Each error or warning during the execution of a PL/SQL block raises an exception. One can distinguish between two types of exceptions:

- System defined exceptions
- User defined exceptions (which must be declared by the user in the declaration part of a block where the exception is used/implemented)

System defined exceptions are always automatically raised whenever corresponding errors or warnings occur. User defined exceptions, in contrast, must be raised explicitly in a sequence of statements using raise <exception name>. After the keyword exception at the end of a block, user defined exception

handling routines are implemented. An implementation has the pattern when <exception name> then <sequence of statements>;

The most common errors that can occur during the execution of PL/SQL programs are handled by system defined exceptions.

Syntax:-

<Exception_name>Exception;

Handling Exceptions:- Exceptions handlers for all the exceptions are written in the exception handling section of a PL/SQL block.

Syntax:-

Exception

When exception_name then Sequence_of_statements1;

When exception_name then Sequence_of_statements2;

When exception_name then Sequence_of_statements3;

End;

Example:

Declare

emp sal EMP.SAL%TYPE; emp no EMP.EMPNO%TYPE;
too_high_sal

exception; begin

select EMPNO, SAL into emp no, emp sal from EMP where ENAME =

"KING"; if emp sal ∗1.05 > 4000 then raise too high sal

else update EMP set SQL . . . end if ;

exception

when NO DATA FOUND – – no tuple selected then rollback;

when too_high_sal then insert into high sal emps values(emp no);

commit; end;

After the keyword when a list of exception names connected with or can be specified. The last when clause in the exception part may contain the exception name others. This introduces the default exception handling routine, for example, a rollback.

If a PL/SQL program is executed from the SQL*Plus shell, exception handling routines may contain statements that display error or warning messages on the screen. For this, the procedure raise application error can be used. This procedure has two parameters <error number> and <message text>.

<error number> is a negative integer defined by the user and must range between -20000 and -20999.

<error message> is a string with a length up to 2048 characters.

The concatenation operator "||" can be used to concatenate single strings to one string. In order to display numeric variables, these variables must be converted to strings using the function to char. If the

procedure raise application error is called from a PL/SQL block, processing the PL/SQL block terminates and all database modifications are undone, that is, an implicit rollback is performed in addition to displaying the error message.

Applications:

Input:

1. Consider table Stud(Roll, Att,Status)

Write a PL/SQL block for following requirement and handle the exceptions.

Roll no. of student will be entered by user. Attendance of roll no. entered by user will be checked in Stud table. If attendance is less than 75% then display the message "Term not granted" and set the status in stud table as "D". Otherwise display message "Term granted" and set the status in stud table as "ND"

2. Write a PL/SQL block for following requirement using user defined exception handling.

The account_master table records the current balance for an account, which is updated whenever, any deposits or withdrawals takes place. If the withdrawal attempted is more than the current balance held in the account. The user defined exception is raised, displaying an appropriate message. Write a PL/SQL block for above requirement using user defined exception handling.

3. 1. Borrower(Roll_no, Name, Date_of_Issue, Name_of_Book, Status)

2. Fine(Roll_no, Date, Amt)

- Accept Roll_no & Name of Book from user.
- Check the number of days (from date of issue), if days are between 15 to 30 then fine amount will be Rs 5per day.
- If no. of days>30, fine will be Rs 50 per day & for days less than 30, Rs. 5 per day.
- After submitting the book, status will change from I to R.

- If condition of fine is true, then details will be stored into fine table.

Also handles the exception by named exception handler or user define exception handler.

Output:

Conclusion:

Hence successfully learned to write PL/SQL code using control structure and exception handling

Outcome:

Able to understand and write code using Control structure and exception handling.

Questions:

1.
2.
3.
4.

DATABASE MANAGEMENT SYSTEM (317523)

Experiment No. 5

Aim:

PL/SQL code block using Cursors (All types: Implicit, Explicit, Cursor FOR Loop, Parameterized Cursor) Problem

Objective:

To write a PL/SQL code block using Cursors (All types: Implicit, Explicit,Cursor FOR Loop, Parameterized Cursor)

Theory:

- **:- CURSOR:-**

For the processing of any SQL statement, database needs to allocate memory. This memory is called context area. The context area is a part of PGA (Process global area) and is allocated on the oracle server.

A cursor is associated with this work area used by ORACLE, for multi row queries. A cursor is a handle or pointer to the context area .The cursor allows to process contents in the context area row by row.

There are two types of cursors.

1) Implicit cursor:-Implicit cursors are defined by ORACLE implicitly. ORACLE defines implicit cursor for every DML statements.

2) Explicit cursor:-These are user-defined cursors which are defined in the declaration section of the PL/SQL block. There are four steps in which the explicit cursor is processed.

1) Declaring a cursor

2) Opening a cursor

3) Fetching rows from an opened cursor

4) Closing cursor

General syntax for CURSOR:- DECLARE

Cursor cursor_name IS select_statement or query;

BEGIN Open cursor_name;

Fetch cursor_name into list_of_variables; Close

cursor_name; END;

Where

1) Cursor_name:-is the name of the cursor.

2) Select_statement:-is the query that defines the set of rows to be processed by the cursor.

3) Open cursor_name:-open the cursor that has been previously declared. When cursor is opened following things happen

i) The active set pointer is set to the first row.

ii) The value of the binding variables are examined.

4) Fetch statement is used to retrieve a row from the selected rows, one at a time, into PL/SQL variables.

5) Close cursor_name:-When all of cursor rows have been retrieved, the cursor should be closed.

Explicit cursor attributes:-

Following are the cursor

attributes

1. %FOUND: - This is Boolean attribute. It returns TRUE if the previous fetch returns a row and false if it doesn‟t.

2. %NOTFOUND:-If fetch returns a row it returns FALSE and TRUE if it doesn‟t. This as the exit condition for the fetch loop; is often used

3. %ISOPEN:-This attribute is used to determine whether or not the associated cursor is open. If so it returns TRUE otherwise FALSE.

4. %ROWCOUNT:-This numeric attribute returns a number of rows fetched by the cursor.

Cursor Fetch Loops

1) **Simple Loop**

Syntax:- LOOP

Fetch cursorname into list of variables;

EXIT WHEN cursorname%NOTFOUND

Sequence_of_statements; END LOOP;

2) **WHILE Loop**

Syntax:-

FETCH cursorname INTO list of variables; WHILE cursorname%FOUND

LOOP Sequence_of_statements;

FETCH cursorname INTO list of variables; END LOOP;

3) **Cursor FOR Loop**

Syntax:

FETCH cursorname INTO list of variables; WHILE cursorname%FOUND LOOP

Sequence_of_statements;

FETCH cursorname INTO list of variables; END LOOP;

FOR variable_name IN cursorname LOOP

-- an implicit fetch is done here.

-- cursorname%NOTFOUND is also implicitly checked.

-- process the fetch records.

Sequence_of_statements; END LOOP;

There are two important things to note about :-

i. Variable_name is not declared in the DECLARE section. This variable is implicitly declared by the PL/SQL compiler.

ii. Type of this variable is cursorname%ROWTYPE.

Implicit Cursors

PL/SQL issues an implicit cursor whenever you execute a SQL statement directly in your code, as long as that code does not employ an explicit cursor. It is called an "implicit" cursor because you, the developer, do not explicitly declare a cursor for the SQL statement.

If you use an implicit cursor, Oracle performs the open, fetches, and close for you automatically; these actions are outside of your programmatic control. You can, however, obtain information about the most recently executed SQL statement by examining the values in the implicit SQL cursor attributes.

PL/SQL employs an implicit cursor for each UPDATE, DELETE, or INSERT statement you execute in a program. You cannot, in other words, execute these statements within an explicit cursor, even if you want to. You have a choice between using an implicit or explicit cursor only when you execute a single-row SELECT statement (a SELECT that returns only one row).

In the following UPDATE statement, which gives everyone in the company a 10% raise, PL/SQL creates an implicit cursor to identify the set of rows in the table which would be affected by the update:

```
UPDATE employee

SET salary = salary * 1.1;
```

The following single-row query calculates and returns the total salary for a department. Once again, PL/SQL creates an implicit cursor for this statement:

```
SELECT SUM (salary) INTO

department_total FROM employee


WHERE department_number = 10;
```

If you have a SELECT statement that returns more than one row, you must use an explicit cursor for that query and then process the rows returned one at a time. PL/SQL does not yet support any kind of array interface between a database table and a composite PL/SQL datatype such as a PL/SQL table.

Drawbacks of Implicit Cursors

Even if your query returns only a single row, you might still decide to use an explicit cursor. The implicit cursor has the following drawbacks:

- It is less efficient than an explicit cursor

- It is more vulnerable to data errors
- It gives you less programmatic control

The following sections explore each of these limitations to the implicit cursor.

Inefficiencies of implicit cursors

An explicit cursor is, at least theoretically, more efficient than an implicit cursor. An implicit cursor executes as a SQL statement and Oracle's SQL is ANSI-standard. ANSI dictates that a single-row query must not only fetch the first record, but must also perform a second fetch to determine if too many rows will be returned by that query (such a situation will RAISE the TOO_MANY_ROWS PL/SQL exception). Thus, an implicit query always performs a minimum of two fetches, while an explicit cursor only needs to perform a single fetch.

This additional fetch is usually not noticeable, and you shouldn't be neurotic about using an implicit cursor for a single-row query (it takes less coding, so the temptation is always there). Look out for indiscriminate use of the implicit cursor in the parts of your application where that cursor will be executed repeatedly. A good example is the Post-Query trigger in the Oracle Forms.

Post-Query fires once for each record retrieved by the query (created from the base table block and the criteria entered by the user). If a query retrieves ten rows, then an additional ten fetches are needed with an implicit query. If you have 25 users on your system all performing a similar query, your server must process 250 additional (unnecessary) fetches against the database. So, while it might be easier to write an implicit query, there are some places in your code where you will want to make that extra effort and go with the explicit cursor.

Vulnerability to data errors

If an implicit SELECT statement returns more than one row, it raises the TOO_MANY_ROWS exception. When this happens, execution in the current block terminates and control is passed to the exception section. Unless you deliberately plan to handle this scenario, use of the implicit cursor is a declaration of faith. You are saying, "I trust that query to always return a single row!"

It may well be that today, with the current data, the query will only return a single row. If the nature of the data ever changes, however, you may find that the SELECT statement which formerly identified a single row now returns several. Your program will raise an exception. Perhaps this is what you will want. On the other hand, perhaps the presence of additional records is inconsequential and should be ignored.

With the implicit query, you cannot easily handle these different possibilities. With an explicit query, your program will be protected against changes in data and will continue to fetch rows without raising exceptions.

Applications:

Input:

Problem Statement:

Implicit Cursor

1. The bank manager has decided to activate all those accounts which were previously marked as inactive for performing no transaction in last 365 days. Write a PL/SQ block (using implicit cursor) to update the status of account, display an approximate message based on the no. of rows affected by the update.

(Use of %FOUND, %NOTFOUND,

%ROWCOUNT) **EXPLICIT CURSOR:**

2. Organization has decided to increase the salary of employees by 10% of existing salary, who are having salary less than average salary of organization, Whenever such salary updates takes place, a record for the same is maintained in the increment_salary table.

EMP (E_no , Salary) increment_salary(E_no , Salary)

3. Write PL/SQL block using explicit cursor for following requirements:

College has decided to mark all those students detained (D) who are having attendance less than 75%. Whenever such update takes place, a record for the same is maintained in the D_Stud table. create table stud21(roll number(4), att number(4), status varchar(1));

create table d_stud(roll number(4), att number(4));

parameterized Cursor

4. Write a PL/SQL block of code using parameterized Cursor, that will merge the data available in the newly created table N_RollCall with the data available in the table O_RollCall. If the data in the first table already exist in the second table then that data should be skipped.

parameterized Cursor

5. Write the PL/SQL block for following requirements using parameterized Cursor:

Consider table EMP(e_no, d_no, Salary), department wise average salary should be inserted into new table dept_salary(d_no, Avg_salary)

EXPLICIT CURSOR: Cursor for loop

1. Write PL/SQL block using explicit cursor: Cursor FOR Loop for following requirements:

College has decided to mark all those students detained (D) who are having attendance less than 75%.

Whenever such update takes place, a record for the same is maintained in the D_Stud table. create table stud21(roll number(4), att number(4), status varchar(1));

create table d_stud(roll number(4), att number(4));

Output:

Conclusion:

Hence successfully learn to implement cursor.

Outcome: Able to write PL/SQL code block using cursors

Questions:

1.

2.

3.

4.

5.

Experiment No. 6

Aim:

Write a program to implement MySQL/Oracle database connectivity with any front-end language to implement Database navigation operations (add, delete, edit etc.)

Objective: Ability to understand database connectivity using database and front end language

Theory:

```
import java.awt.*; import java.awt.event.*;

import java.sql.Connection; import java.sql.DriverManager; import java.sql.ResultSet; import

java.sql.Statement; import javax.swing.*;

public class student extends JFrame implements ActionListener{

JFrame f;
JLabel l1, l2,l3,l4; JTextField t1, t2,t3; JButton b1, b2, b3, b4, b5; Connection c;
Statement s; ResultSet r; student ()
{try{
f=new JFrame("Student Form"); f.setLayout(null);f.setVisible(true); f.setSize(700, 500);
l4=new JLabel("Student Management System");
//l4.setBounds(100,01,250,250); l4.setBounds(100, 30, 400, 30);
f.add(l4); l4.setForeground(Color.blue);
l4.setFont(new Font("Serif", Font.BOLD,
30)); l1=new JLabel("Stud_RollNo");

l1.setBounds(50, 70, 100, 50);f.add(l1); l2=new JLabel("Stud_Name"); l2.setBounds(50, 120, 100, 50);
f.add(l2);
l3=new JLabel("Stud_Dept"); l3.setBounds(50, 170, 100, 50); f.add(l3);
t1=new JTextField(); t1.setBounds(150, 90, 100, 30); f.add(t1);
t2=new JTextField(); t2.setBounds(150, 140, 100, 30); f.add(t2);t3=new JTextField();
t3.setBounds(150, 190, 100, 30); f.add(t3);
b1= new JButton("ADD"); b1.setBounds(200, 300, 75, 50); f.add(b1); b1.addActionListener(this); b2=
new JButton("EDIT"); b2.setBounds(300, 300, 75, 50); f.add(b2); b2.addActionListener(this);
b3= new JButton("DELETE");b3.setBounds(400, 300, 75, 50); f.add(b3);
b3.addActionListener(this); b5= new JButton("EXIT"); b5.setBounds(500, 300, 75, 50);
f.add(b5); b5.addActionListener(this);

Class.forName("com.mysql.jdbc.Driver"); c=DriverManager.getConnection("jdbc:mysql://loca
lhost:3306/info","root","root");s=c.createStatement();
}catch(Exception e){System.out.println(e);}
}//ends INS Constructor
public void actionPerformed(ActionEvent ae){ try{
if(ae.getSource()==b1){String s1="INSERT INTO result(stud_RollNo,stud_Name,stud_Dept)
VALUES("+t1.getText()+",'"+t2.getText()
```

```
+"','"+t3.getText() + "')";
System.out.println(s1); s.executeUpdate(s1);
r=s.executeQuery("SELECT * FROM result"); t1.setText("");

t2.setText("");
t3.setText("");
}else if(ae.getSource()==b2){
String s2="UPDATE user1 SET stud_Name='"+t2.getText()+"' WHERE stud_RollNo="+t1.getText();
System.out.println(s2); s.executeUpdate(s2);
r=s.executeQuery("SELECT * FROM result"); t1.setText("");
t2.setText("");t3.setText("");
}else if(ae.getSource()==b3){
String s3="DELETE FROM result WHERE stud_RollNo="+t1.getText(); System.out.println(s3);
s.executeUpdate(s3); r=s.executeQuery("SELECT * FROM result"); t1.setText("");
t2.setText("");
t3.setText("");}else if(ae.getSource()==b5){System.exit(0); }
}catch(Exception e){System.out.println(e);}
}
public static void main(String args[]){ new student();
}
}
```

Applications:

Input:

Problem Statement:

Write a program to implement MySQL/Oracle database connectivity with any front end language to implement Database navigation operations (add, delete, edit etc.)

Output:

```
sl2-pc5@sl2pc5-HP-Compaq-4000-Pro-SFF-PC:~$ mysql -u root -p
Enter password:
Welcome to the MySQL monitor. Commands end with
; or \g.
Your MySQL connection id is 42
Server version: 5.5.61-0ubuntu0.14.04.1 (Ubuntu) Copyright (c) 2000, 2018, Oracle and/or its affiliates.
All rights reserved.
Oracle is a registered trademark of Oracle
Corporation and/or itsaffiliates. Other names may be trademarks of their
respective owners.
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement. mysql> create database info;
Query OK, 1 row affected (0.03 sec)

mysql> use info;
Database changed
```

```
mysql> create table result (stud_RollNo int,stud_Name varchar(20),stud_Dept varchar(20));
Query OK, 0 rows affected (0.08 sec) mysql> select *from result;
+-------------+-----------+        +| stud_RollNo | stud_Name | stud_Dept |


    +       +       +       +
    |
    1 | abc
    | comp
    |
    +       +       +       + 1 row in set (0.00 sec)
//ADD DATA
mysql> select *from result;
+       +       +       +
| stud_RollNo | stud_Name | stud_Dept |
+       +       +       +
|
1       | abc
| comp
|
|
2       | harsha
| comp
|
|
3       | tej
| comp
|
|
4       | rina
| mech
|
+-------------+-----------+        +4 rows in set (0.00 sec)
//DELETE DATA
mysql> select *from result;
+-------------+-----------+        +| stud_RollNo | stud_Name | stud_Dept |
+       +       +       +
|
2       | harsha
| comp
|

|
3       | tej
| comp
|
|
4       | rina
```

| mech
|
+ + + + 3 rows in set (0.00 sec)

Conclusion: successfully learned to write code for database connectivity

Outcome: Able to understood database connectivity using database and front end language

DATABASE MANAGEMENT SYSTEM (317523)

DATABASE MANAGEMENT SYSTEM (317523)

DATABASE MANAGEMENT SYSTEM (317523)

Questions:

1.

2.

3.

4.

5.

4. Appendix

DATABASE MANAGEMENT SYSTEM (317523)

DATABASE MANAGEMENT SYSTEM (317523)

DATABASE MANAGEMENT SYSTEM (317523)

DATABASE MANAGEMENT SYSTEM (317523)

.

Experiment No. 5

Aim:

Write a program for pre-processing of a text document such as stop word removal, stemming.

Objective:

To Study:

1. Pre-processing of text documents
2. Removing Stop Words with NLTK
3. Performing the Stopwords operations in a file

Theory:

Applications:

1. To eliminate unimportant words
2. To allow applications to focus on the important words
3. To drop common words

Input:

Output:

Conclusion:

Thus we have implemented pre-processing of a text document such as stop word removal, stemming**.**

Outcome:

Upon completion of this experiment, students will be able to:

Experiment level outcome (ELO1): Perform pre-processing of a text document such as stop word removal, stemming using NLTK

Questions:

1.

2.

3.

4.

5.

4.1.Experiment No. 6

Aim:

Write a program for pre-processing of a text document such as stop word removal, stemming.

Objective:

To Study:

1. Pre-processing of text documents
2. Removing Stop Words with NLTK
3. Performing the Stopwords operations in a file

Theory:

Applications:

1. To eliminate unimportant words
2. To allow applications to focus on the important words
3. To drop common words

Input:

Output:

Conclusion:

Thus we have implemented pre-processing of a text document such as stop word removal, stemming**.**

Outcome:

Upon completion of this experiment, students will be able to:

Experiment level outcome (ELO1): Perform pre-processing of a text document such as stop word removal, stemming using NLTK

Questions:

1.

2.

3.

4.

5.

5. Appendix