ASHA M. TARSADIA INSTITUTE OF COMPUTER SCIENCE AND TECHNOLOGY

CERTIFICATE

This is to certify that Mr. /Ms._____,

Enrollment No: _____ of B.Tech. Computer Science and Engineering 3rd semester has satisfactorily completed his/her laboratory work of Database Management System during regular term in academic year 2023-24.

Date of Submission: _____

Department Authority

Ms. Snehal Kathale
Subject Incharge
Assistant Professor
Department of Computer Science
and Engineering

Institute Stamp

Uka Tarsadia University
Asha M. Tarsadia Institute of Computer Science and Technology

INDEX

Sr. No.	Practical	Submission Date	Sign
1	To implement Basic SQL commands and to access & modify Data using SQL. Create and populate database using Data Definition Language (DDL) and DML Commands		
2	Implement DDL and DML queries with different clauses.		
3	To implement aggregate functions.		
4	To implement Integrity Constraints. Queries (along with sub-Queries)		
5	Queries using: **String functions** (Concatenation, lpad, rpad, ltrim, rtrim, lower, upper, initcap, length, substr and instr, ELT(), Char_Length(), Format(), Find_In_Set, Oct(), Reverse(), Repeat(), Ascii()) **Date functions**:(Sysdate, next_day, add_months, last_day, months_between, to_char, to_date, PERIOD_DIFF) **Numeric functions**: Abs (), power (), sqrt (), greatest (), least (), round (), mod (). **Time functions** (Localtime, Minute(*datetime*), Microsecond)		
6	To implement Joins.		
7	To Perform Simple queries, string manipulation operations implement group by having.		
8	Identify the case study and detail statement of problem. Design an Entity-Relationship (ER) / Extended Entity-Relationship (EER) Model.		
9	Implement queries related to order by and having clause.		
10	Create a simple PL/SQL program which includes declaration section, executable section and exception –Handling section (Ex. Student marks can be selected from the table and printed for those who secured first class and an exception can be raised if no records were found) ii. Insert data into student table and use COMMIT, ROLLBACK and SAVEPOINT in PL/SQL block.		
11	Implement set operations, case statement and view queries.		
12	Develop Programs using BEFORE and AFTER Triggers, Row and Statement Triggers and INSTEAD OF Triggers		
13	Create a table and perform the search operation on table using indexing and non-indexing techniques.		
14	Programs development using creation of procedures, passing		

	parameters IN and OUT of PROCEDURES		
15	To implement B-trees/B+ trees and Indexing.		

Practical No:- 12

Aim: Develop Programs using BEFORE and AFTER Triggers, Row and Statement Triggers and INSTEAD OF Triggers

Theory:-

Triggers in SQL are special types of stored procedures that are automatically executed (or "triggered") in response to certain events or actions that occur in a database. They provide a way to automate tasks, enforce data integrity, and add business logic to your database.

BEFORE Triggers:

- A BEFORE trigger is executed before the triggering SQL statement.
- It can be used to validate or modify the data being inserted, updated, or deleted.
- If a BEFORE trigger raises an error, the triggering SQL statement will be aborted.

AFTER Triggers:

- An AFTER trigger is executed after the triggering SQL statement.
- It is often used for tasks that need to be performed after a change has been made, such as logging or auditing.
- Errors in AFTER triggers do not affect the outcome of the triggering SQL statement.

Row-level Triggers:

- Row-level triggers are triggered once for each row affected by a SQL statement.
- They allow you to access and manipulate the data in the affected rows individually.
- They are useful for enforcing constraints or maintaining data consistency on a per-row basis.

Statement-level Triggers:

- Statement-level triggers are triggered once for each SQL statement, regardless of the number of rows affected.
- They are typically used for auditing or logging purposes.
- They provide a broader view of the entire operation.

INSTEAD OF Triggers:

- INSTEAD OF triggers are used on views and allow you to perform custom actions instead of the default INSERT, UPDATE, or DELETE operations on the view.

- They are helpful when you need to implement complex logic for operations on views.

Query 1: Trigger will display the salary difference between the old values and new values.

SQL Worksheet

```
1  UPDATE Employee
2  SET Salary = 60000
3  WHERE EmployeeID = 2;
4
5  select * from employee;
```

```
2 row(s) updated.
Old Salary: 60000
New Salary: 60000
Salary Difference: 0
Old Salary: 60000
New Salary: 60000
Salary Difference: 0
```

EMPLOYEEID	FIRSTNAME	LASTNAME	SALARY
1	John	Doe	60000
2	Jane	Smith	60000
1	John	Doe	60000
1	John	Doe	60000
2	Jane	Smith	60000

Query 2: Stop the transaction if the quantity entered (insert) by the user exceeds 1000.

SQL Worksheet

```
1  /*202203103510124*/
2  CREATE TRIGGER QuantityLimitTrigger
3  BEFORE INSERT ON Orders
4  FOR EACH ROW
5  BEGIN
6      IF :NEW.Quantity > 1000 THEN
7          RAISE_APPLICATION_ERROR(-20001, 'Quantity limit exceeded (maximum is 1000). Transaction is not allowed.');
8      END IF;
9  END;
10 /
```

```
Trigger created.
```

SQL Worksheet Clear Find

```
1  /*202203103510124*/
2  INSERT INTO Orders (OrderID, Product, Quantity)
3  VALUES (1, 'Product1', 1200);
```

```
ORA-20001: Quantity limit exceeded (maximum is 1000). Transaction is not allowed. ORA-06512: at "SQL_UOOYHXFGTBCEAUBVKRJOTWVLB.QUANTITYLIMITTRIGGER", line 3
ORA-06512: at "SYS.DBMS_SQL", line 1721

More Details: https://docs.oracle.com/error-help/db/ora-20001
```

Query 3: Maintain Log Table with use of trigger.

SQL Worksheet

```
1  /*202203103510124*/
2  CREATE OR REPLACE TRIGGER EmployeeLogBeforeTrigger
3  BEFORE UPDATE ON Employee
4  FOR EACH ROW
5  BEGIN
6      INSERT INTO EmployeeLog (LogID, EmployeeID, LogMessage)
7      VALUES (SEQ_EMPLOYEELOG.NEXTVAL, :OLD.EmployeeID, 'Salary updated from ' || :OLD.Salary || ' to ' || :NEW.Salary);
8  END;
9  /
10
```

```
Trigger created.
```

SQL Worksheet

```
1  /*202203103510124*/
2
3  UPDATE Employee SET Salary = 2000 WHERE FirstName = 'John';
```

```
5 row(s) updated.
Old Salary: 60000
New Salary: 2000
Salary Difference: -58000
Old Salary: 60000
New Salary: 2000
Salary Difference: -58000
Old Salary: 50000
New Salary: 2000
Salary Difference: -48000
Old Salary: 50000
New Salary: 2000
Salary Difference: -48000
Old Salary: 60000
New Salary: 2000
Salary Difference: -58000
```

Conclusion :

When implementing triggers, it's important to carefully plan and test them to ensure that they achieve their intended purpose without introducing unintended side effects or performance issues in your database. Triggers can be powerful, but they should be used judiciously and with a clear understanding of their impact on the database.

Practical No. :- 13

Aim :-Create a table and perform the search operation on table using indexing and non-indexing techniques.

Theory :-

- Definition: Indexing is a database optimization technique used to speed up data retrieval operations, such as searching and querying.
- How it works: An index is a data structure that contains a copy of specific columns from a table, organized in a way that makes it faster to find rows that match a certain value. It acts as a roadmap to the data in the table.

Types of Indexes:

- Clustered Index: Determines the physical order of data in a table. There can be only one clustered index per table.
- Non-clustered Index: Creates a separate data structure that contains a copy of indexed columns and a pointer to the actual rows. Multiple non-clustered indexes can be created per table.
- Benefits: Indexing speeds up data retrieval, reduces the need for full table scans, and improves query performance.

Non-Indexing:

- Definition: Non-indexing refers to not creating indexes on a table or specific columns.
- How it works: When no indexes are created, the database management system has to scan the entire table to find rows that match a search condition. This can be slower for large datasets.
- Impact: Search operations without indexing can be resource-intensive, slower, and less efficient. They may lead to increased query execution times.

Index :

Using index and Without index :

```
mysql> /*202203103510124*/
mysql> -- Search for an employee without an index
mysql> SELECT * FROM Employee WHERE LastName = 'Johnson';
+------------+-----------+----------+------------+----------+
| EmployeeID | FirstName | LastName | Department | Salary   |
+------------+-----------+----------+------------+----------+
|          1 | Alice     | Johnson  | HR         | 50000.00 |
+------------+-----------+----------+------------+----------+
1 row in set (0.00 sec)

mysql> CREATE INDEX idx_empLastName ON Employee (LastName);
Query OK, 0 rows affected (0.03 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> /*202203103510124*/
mysql> -- Search for an employee using an index
mysql> SELECT * FROM Employee WHERE LastName = 'Johnson';
+------------+-----------+----------+------------+----------+
| EmployeeID | FirstName | LastName | Department | Salary   |
+------------+-----------+----------+------------+----------+
|          1 | Alice     | Johnson  | HR         | 50000.00 |
+------------+-----------+----------+------------+----------+
1 row in set (0.00 sec)
```

Conclusion :

- Indexing is a critical technique in database management to optimize search operations.
- Indexes create data structures that act as a roadmap to the data, making it faster to locate specific rows.
- Non-indexed searches may result in slower query performance, especially in tables with a large number of rows.
- The decision to use indexing should be based on the specific requirements of the database and the expected access patterns. It's important to balance the benefits of indexing with the additional storage space and potential impacts on insert and update operations.

Practical No. :- 14

Aim :-Programs development using creation of procedures, passing parameters IN and OUT of

PROCEDURES

Theory :-

In database management, stored procedures are a powerful feature that allows you to encapsulate a sequence of SQL statements into a single unit. They provide several advantages, including code reusability, improved security, and enhanced performance. Procedures can accept input parameters (IN), process data, and return output parameters (OUT) or result sets.

IN Parameters:

- IN parameters are used to pass values into a procedure. These values can be used within the procedure's SQL statements.
- IN parameters are typically used for input data or conditions.
- They help make procedures more flexible and reusable, as you can customize the behavior of the procedure based on the input values.

OUT Parameters:

- OUT parameters are used to return values or data from a procedure to the calling program.
- They allow you to obtain results or data calculated within the procedure.
- OUT parameters are useful when you want to retrieve specific data generated by the procedure.

Query 1: Create one Stored Procedure to Insert, Delete and Update into one table.

```
mysql> /*202203103510124*/
mysql> DELIMITER //
mysql> CREATE PROCEDURE ModifyProduct4(
    ->     IN action VARCHAR(10),
    ->     IN productID INT,
    ->     IN productName VARCHAR(255),
    ->     IN price DECIMAL(10, 2)
    -> )
    -> BEGIN
    ->     IF action = 'INSERT' THEN
    ->         INSERT INTO Product3 (ProductID, ProductName, Price) VALUES (productID, productName, price);
    ->     ELSEIF action = 'DELETE' THEN
    ->         DELETE FROM Product3 WHERE ProductID = productID;
    ->     ELSEIF action = 'UPDATE' THEN
    ->         UPDATE Product3 SET ProductName = productName, Price = price WHERE ProductID = productID;
    ->     END IF;
    -> END;
    -> //
Query OK, 0 rows affected (0.00 sec)

mysql> DELIMITER ;
mysql> CALL ModifyProduct4('INSERT', 101, 'New Product', 49.99);
ERROR 1062 (23000): Duplicate entry '101' for key 'product3.PRIMARY'
mysql> CALL ModifyProduct4('INSERT', 103, 'New Product2', 49.99);
Query OK, 1 row affected (0.00 sec)

mysql> CALL ModifyProduct4('UPDATE', 1012, 'Updated Product2', 59.99);
Query OK, 2 rows affected (0.00 sec)

mysql> CALL ModifyProduct4('DELETE', 101, '', 0);
Query OK, 2 rows affected (0.00 sec)
```

Query 2: Create one Stored Procedure to increment salary with salary range limit.

```
mysql> /*202203103510124*/
mysql> CREATE TABLE SalaryIncrement (
    ->     EmployeeID INT PRIMARY KEY,
    ->     EmployeeName VARCHAR(255),
    ->     Salary DECIMAL(10, 2)
    -> );
Query OK, 0 rows affected (0.02 sec)

mysql> /*202203103510124*/
mysql> DELIMITER //
mysql> CREATE PROCEDURE IncrementSalary(
    ->     IN employeeID INT,
    ->     IN salaryIncrement DECIMAL(10, 2)
    -> )
    -> BEGIN
    ->     DECLARE currentSalary DECIMAL(10, 2);
    ->
    ->     SELECT Salary INTO currentSalary FROM SalaryIncrement WHERE EmployeeID = employeeID;
    ->
    ->     IF currentSalary + salaryIncrement <= 100000 THEN
    ->         UPDATE SalaryIncrement SET Salary = currentSalary + salaryIncrement WHERE EmployeeID = employeeID;
    ->     ELSE
    ->         SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Salary exceeds the limit (100000).';
    ->     END IF;
    -> END;
    -> //
Query OK, 0 rows affected (0.01 sec)
```

```
mysql> -- Increment salary for an employee
mysql> CALL IncrementSalary(201, 5000);
ERROR 1644 (45000): Salary exceeds the limit (100000).
mysql>
```

Query 3: Create one Stored Procedure to find largest price with that product name.

```
mysql> /*202203103510124*/
mysql> DELIMITER //
mysql> CREATE PROCEDURE FindLargestPrice5(
    ->     IN productName VARCHAR(255)
    -> )
    -> BEGIN
    ->     SELECT MAX(Price) FROM Product3 WHERE ProductName = productName;
    -> END;
    -> //
Query OK, 0 rows affected (0.00 sec)

mysql> DELIMITER ;
mysql>
mysql>
mysql> call FindLargestPrice5('dish');
+------------+
| MAX(Price) |
+------------+
|     999.99 |
+------------+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)
```

Conclusion :

Stored procedures with input (IN) and output (OUT) parameters are valuable tools in database development. They allow you to encapsulate and reuse complex SQL logic, improving code maintainability and security. IN parameters enable you to pass values and conditions into the procedure, making it adaptable to various scenarios. OUT parameters provide a means to return calculated results or data back to the calling program, enhancing the procedure's versatility. When using procedures with IN and OUT parameters, it's crucial to design them efficiently, taking into consideration the specific needs of your application. Careful planning and proper use of IN and OUT parameters can lead to more maintainable and efficient database code.

Enrollment No. : 202203103510405

Practical No. :- 15

Aim :- To implement B-trees/B+ trees and Indexing.

Theory :-

B-trees and B+ trees are advanced data structures used in computer science and databases to efficiently store and manage large sets of data. They are particularly valuable for indexing, which enhances the speed and efficiency of data retrieval operations in databases. Here's a theoretical explanation of these concepts:

B-tree:

- A B-tree is a self-balancing tree data structure that maintains sorted data and allows for efficient insertion, deletion, and search operations. It is characterized by a balanced structure with a branching factor, typically denoted as "b." Each node in a B-tree can have a variable number of keys and pointers to child nodes, and the tree is balanced to ensure consistent depth. This balance is maintained through redistributing keys between nodes and splitting or merging nodes when necessary.

B+ tree:

- A B+ tree is a specific type of B-tree designed for use in databases. Unlike regular B-trees, B+ trees store data only in leaf nodes, while non-leaf nodes contain keys for efficient traversal. The B+ tree's key properties include a balanced structure, sorted order of keys in leaf nodes, and linked leaf nodes for quick range queries. B+ trees are well-suited for indexing in databases due to their predictable and efficient structure.

Index:

- An index is a database structure that provides a fast and efficient way to look up records in a table based on the values in one or more columns. It serves as a data structure that maps values to their corresponding rows in a table. Indexes are created on specific columns in database tables to optimize the retrieval of records. They can be built on single or multiple columns, allowing for quicker access to data.

Advantages of Indexing:

- Indexing improves the speed of data retrieval operations in a database. When a query involves a column with an index, the database can use the index to quickly locate the rows that meet the query's criteria, reducing the need for full table scans. This results in significant performance gains, especially for large datasets.

Implementation:

- **Creating B-trees and B+ trees:** Implementing B-trees and B+ trees involves organizing data in a hierarchical structure with balanced nodes. The branching factor (b) determines the maximum number of keys a node can hold. Insertion and deletion operations require maintaining balance by redistributing keys or splitting/merging nodes as needed.

- **Creating Indexes:** Indexes are created using SQL commands in database management systems. The database engine automatically manages the index structure as data is inserted, updated, or deleted. Common types of indexes include B-tree, B+ tree, and Hash indexes, depending on the DBMS.

Use Cases:

- B-trees and B+ trees are widely used in databases for indexing on primary keys, foreign keys, and other frequently queried columns.

- They are essential for optimizing queries involving range searches, sorting, and filtering.

Creating a B-Tree Index:

```
mysql> create table products1 (
    -> product_id int auto_increment primary key ,
    -> product_name varchar(50) ,
    -> price decimal (10, 2)
    -> ) ;
Query OK, 0 rows affected (0.01 sec)
```

```
mysql> insert into products1 (product_name , price) values
    -> ( 'product 1', 10.00) ,
    -> ( 'product 2', 20.00) ,
    -> ( 'product 3', 30.00) ,
    -> ( 'product 4', 40.00) ,
    -> ( 'product 5', 50.00) ;
Query OK, 5 rows affected (0.00 sec)
Records: 5  Duplicates: 0  Warnings: 0
```

```
mysql> create index idx_product_name on products1 (product_name) ;
Query OK, 0 rows affected (0.03 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Using the Index:

```
mysql> explain select * from products1 where product_name = 'product 3';
+----+-------------+----------+------------+------+------------------+------------------+---------+-------+------+----------+-------+
| id | select_type | table    | partitions | type | possible_keys    | key              | key_len | ref   | rows | filtered | Extra |
+----+-------------+----------+------------+------+------------------+------------------+---------+-------+------+----------+-------+
|  1 | SIMPLE      | products1 | NULL       | ref  | idx_product_name | idx_product_name | 203     | const |    1 |   100.00 | NULL  |
+----+-------------+----------+------------+------+------------------+------------------+---------+-------+------+----------+-------+
1 row in set, 1 warning (0.00 sec)
```

```
mysql> select * from products1 where product_name = 'product 3';
+------------+--------------+-------+
| product_id | product_name | price |
+------------+--------------+-------+
|          3 | product 3    | 30.00 |
+------------+--------------+-------+
1 row in set (0.00 sec)
```

Creating a B+ Tree Index (Composite Index):

```
mysql> /*2022031035101124*/
mysql> create table students1 (
    -> student_id int auto_increment primary key ,
    -> first_name varchar(50) ,
    -> last_name varchar(50) ,
    -> age int
    -> ) ;
Query OK, 0 rows affected (0.01 sec)

mysql> insert into students1 (first_name , last_name , age)
    -> values
    -> ('john ', 'doe', 25),
    -> ('jane ', 'smith', 22) ,
    -> ('alice', 'johnson', 24) ,
    -> ('bob ', 'brown ', 28) ,
    -> ('ella', 'wilson', 23) ;
Query OK, 5 rows affected (0.01 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql> create index idx_age on students1 (age) ;
Query OK, 0 rows affected (0.03 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Using the Index:

```
mysql> explain select * from students1 where age = 23;
+----+-------------+----------+------------+------+---------------+---------+---------+-------+------+----------+-------+
| id | select_type | table    | partitions | type | possible_keys | key     | key_len | ref   | rows | filtered | Extra |
+----+-------------+----------+------------+------+---------------+---------+---------+-------+------+----------+-------+
|  1 | SIMPLE      | students1 | NULL       | ref  | idx_age       | idx_age | 5       | const |    1 |   100.00 | NULL  |
+----+-------------+----------+------------+------+---------------+---------+---------+-------+------+----------+-------+
1 row in set, 1 warning (0.00 sec)

mysql> select * from students1 where age = 23;
+------------+------------+-----------+------+
| student_id | first_name | last_name | age  |
+------------+------------+-----------+------+
|          5 | ella       | wilson    |   23 |
+------------+------------+-----------+------+
1 row in set (0.00 sec)
```

Conclusion: The implementation of B-trees, B+ trees, and indexing is fundamental to the performance and efficiency of modern database systems. They enable rapid data retrieval, sorting, and querying, making them invaluable tools for managing and accessing large datasets in various applications, including databases, file systems, and search engines.