**Veermata Jijabai Technological Institute**

**Matunga East, Mumbai - 400019**



**Department of Computer Engineering and Information Technology**

**Academic Year: 2016-2017**

**Lab Manual**

**Subject: Database Management System Lab**

**Semester: IV**

**S.Y. B.TECH IT**

**By**

**Dr. V. B. Nikam**

|  |  |  |  |
| --- | --- | --- | --- |
| **Experiment No.** | | **Name of Experiment** | **No of Labs** |
| **A** | | **Design Database** |  |
|  | 1 | Defining Problem Statement | 1 |
|  | 2 | Data Modeling for problem defined in exp1   * ER/EER Diagram * Optimization of ER/EER Diagram | 2 |
|  | 3 | Creation and Normalization of database schema from ER Diagram | 2 |
|  | 4 | Building Database using DDL Statements | 1 |
|  | 5 | Performing different DML operations on Created Databases | 1 |
| **B** | | **Database Operations: SQL and PL/SQL** |  |
|  | 6 | Performing query operations using SQL statement | 1 |
|  | 7 | Writing PL/SQL programs, procedures, triggers | 1 |
| **C** | | **Database Connectivity, GUI and Report** |  |
|  | 8 | Development of GUI for implementing the Problem statement and perform database operations (use database connection string). | 2 |
|  | 9 | Developing GUI/Forms for the problem statement chosen & performing operations through GUI. | 2 |
|  | 10 | Generating information reports for data in database. | 1 |

**Lab Plan**

**List of Experiments**

**Experiment 1: Defining Problem Statement**

**Objective:** Define a problem statement for database real life scenario.

**Theory & Concepts:**

While defining problem statement consider following points

* Should have transactions

There should be different operations performed on the database such as different data insertion, updation and other data processing.

* Should have multiple entities involved

An entity is a “thing” or “object” in the real world that is distinguishable from all other objects. So in the case study there should be multiple objects that will be required to store in database.

* Complex level entity should be involved.

Case study should involve complex objects which will need complex query processing.

**Implementation:**

**Problem statement**

**Results:**

**Experiment 2: Data Modeling for problem defined in exp1**

* ER/EER Diagram
* Optimization of ER/EER Diagram

**Objective:** Design and draw an ER diagram using standard notations for your problem definition and optimize the ER diagram.

**Theory & Concepts:**

**ER Diagram:**

Entity–relationship model (ER model) is a data model for describing a database in an abstract way. The entity-relationship (E-R) data model perceives the real world as consisting of basic objects, called *entities*, and *relationships* among these objects.

The E-R data model employs three basic notions: entity sets, relationship sets, and attributes.

Entity Set:

An entity is a “thing” or “object” in the real world that is distinguishable from all other objects. An entity has a set of properties, and the values for some set of properties may uniquely identify an entity. An **entity set** is a set of entities of the same type that share the same properties, or attributes.

An entity is represented by a set of **attributes**. Attributes are descriptive properties possessed by each member of an entity set.

An attribute, as used in the E-R model, can be characterized by the following attribute types.

* Simple and composite attributes
* Single-valued and multi valued attributes.
* Derived attribute.

Keys:

The attribute values of an entity must be such that they can *uniquely identify* the entity. There are different types of keys:

* Super key
* Candidate Key
* Primary Key

Relationship Set:

A relationship captures how entities are related to one another. A relationship is an association among several entities. A relationship set is a set of relationships of the same type.

An E-R enterprise schema may define certain constraints known as Mapping cardinalities. **Mapping cardinalities**, or cardinality ratios, express the number of entities to which another entity can be associated via a relationship set.

For a binary relationship the mapping cardinality must be one of the following:

* **One to one**.
* **One to many**
* **Many to one**
* **Many to many**

Weak Entity Set:

Entity types that do not have key attributes of their own are called weak entity types.

**Extended E-R Features**

Although the basic E-R concepts can model most database features, some aspects of a database may be more aptly expressed by certain extensions to the basic E-R model. This includes the extended E-R features of specialization, generalization, higher- and lower-level entity sets, attribute inheritance, and aggregation.

Specialization

An entity set may include sub-groupings of entities that are distinct in some way from other entities in the set. For instance, a subset of entities within an entity set may have attributes that are not shared by all the entities in the entity set.

Generalization

The refinement from an initial entity set into successive levels of entity sub-groupings represents a top-down design process in which distinctions are made explicit. The design process may also proceed in a bottom-up manner, in which multiple entity sets are synthesized into a higher-level entity set on the basis of common features.

**Attribute Inheritance**

A crucial property of the higher- and lower-level entities created by specialization and generalization is attribute inheritance. The attributes of the higher-level entity sets are said to be inherited by the lower-level entity sets.

**Constraints on Generalizations**

To model an enterprise more accurately, the database designer may choose to place certain constraints on a particular generalization. One type of constraint involves determining which entities can be members of a given lower-level entity set. Such membership may be one of the following:

* Condition-defined.
* User-defined.

A second type of constraint relates to whether or not entities may belong to more than one lower-level entity set within a single generalization. The lower-level entity sets may be one of the following:

* Disjoint
* Overlapping

A final constraint, the completeness constraint on a generalization or specialization, specifies whether or not an entity in the higher-level entity set must belong to at least one of the lower-level entity sets within the generalization/specialization. This constraint may be one of the following:

* Total generalization or specialization
* Partial generalization or specialization

**Aggregation**

One limitation of the E-R model is that it cannot express relationships among relationships. The best way to model a situation such as the one just described is to use aggregation. **Aggregation** is an abstraction through which relationships are treated as higher level entities.

**Implementation:**

**Results:**

**Experiment 3: Creation and Normalization of database schema from the ER/EER Data Model**

**Objective:** Create relational schema from the ER diagram using mapping rules.

**Theory & Concepts:**

Mapping ER/EER to Database schema involves following steps:

1) Mapping regular entity types to tables

2) Mapping weak entity types

3) Mapping binary 1:1 relationship

4) Mapping binary 1:N relationship

5) Mapping binary M:N relationship

6) Mapping multi-valued attributes

7) Mapping N-ary relationship

8) Mapping generalization/specialization

9) Mapping shared subclasses/multiple inheritance

10)Mapping categories/union types:

**Normalization of Database**

Normalization is a systematic approach of decomposing tables to eliminate data redundancy and undesirable characteristics like Insertion, Update and Deletion Anamolies. It is a two-step process that puts data into tabular form by removing duplicated data from the relation tables.

**Normalization Techniques**

Normalization rule are divided into following normal form.

First Normal Form (1NF)

A row of data cannot contain repeating group of data i.e. each column must have a unique value. Each row of data must have a unique identifier i.e. Primary key.

Second Normal Form (2NF)

A table to be normalized to Second Normal Form should meet all the needs of First Normal Form and there must not be any partial dependency of any column on primary key.

Third Normal Form (3NF)

Third Normal form applies that every non-prime attribute of table must be dependent on primary key. The transitive functional dependency should be removed from the table. The table must be in Second Normal form.

Boyce and Codd Normal Form (BCNF)

Boyce and Codd Normal Form is a higher version of the Third Normal form. This form deals with certain type of anamoly that is not handled by 3NF. A 3NF table which does not have multiple overlapping candidate keys is said to be in BCNF.

**Implementation:**

**Results:**

**Experiment 4: Building Database using DDL Statements**

**Objective:** Create Database Tables, constraints on the Tables, Indices etc using SQL DDL statements and applying them all for the Data Model created for the problem statement chose.

**Theory & Concepts:**

**Data Definition Language**

The SQL DDL provides commands for defining relation schemas, deleting relations, and modifying relation schemas.

Typically a database administrator is responsible for using DDL statements or production databases in a large database system. The commands used are:

* Create - It is used to create a table.
* Alter - This command is used to add a new column, modify the existing column definition and to include or drop integrity constraint.
* Drop - It will delete the table structure provided the table should be empty.
* Truncate - If there is no further use of records stored in a table and the structure has to be retained and then the records alone can be deleted.
* Desc - This is used to view the structure of the table.

Syntax – Create Table

create table <table name>

{

fieldname-1datatype constraints if any,

fieldname-2datatype constraints if any,

…….

fieldname-ndatatype constraints if any,

};

create table <table name> as

(

select(att-list) from <existing table name>

);

Syntax – Alter Table

alter table <table name> add/modify

(

fieldname-1datatype,

fieldname-2datatype,

fieldname-ndatatype

);

alter table drop column column name;

Syntax – Drop Table

droptable <table name>

**Implementation:**

**Results:**

**Experiment 5: Performing different DML operations on created Databases**

**Objective:** Perform different data manipulations and data operations on database using DML. understand and perform operations on VIEWS.

Use SQL DML statements such as INSERT, UPDATE, and DELETE to insert the data into tables and to update/delete the data inserted into/from tables.

**Theory & Concepts:**

**Data Manipulation Language**

DML commands are the most frequently used SQL commands and is used to query and manipulate the

existing database objects. Some of the commands are

1. Insert
2. Select
3. Update
4. Delete

**INSERT:** This is used to add one or more rows to a table. The values are separated by commas and the data types char and date are enclosed in apostrophes. The values must be entered in the same order as they are defined.

Inserting a single row into a table:

insert into <table name> values(fieldvalue-1,fieldvalue-2,…,fieldvalue-n);

Inserting more than one record using a single insert command:

insert into <table name> values(&fieldname-1,&fieldname-2,…&fieldname-n);

Skipping the fields while inserting:

insert into <tablename(coln names to which datas to b inserted)> values (list of values);

Other way is to give null while passing the values.

insert into <table name>(select(att\_list) from <existing table name>);

**SELECT:** - It is used to retrieve information from the table. it is generally referred to as querying the table. We can either display all columns in a table or only specify column from the table.

SELECT(att\_list) FROM <table name> [WHERE <condition/expression>];

Retrieval of all columns from a table:

Select \* from tablename; // This query selects all rows from the table.

Retrieval of specific columns from a table:

It retrieves the specified columns from the table.

Syntax: Select column\_name1, …..,column\_namen from table name;

Elimination of duplicates from the select clause:

It prevents retrieving the duplicated values .*Distinct*keyword is to be used.

Syntax: Select DISTINCT col1, col2 from table name;

Select command with where clause:

To select specific rows from a table we include ‘where’ clause inthe select command. It can appear only after the ‘from’ clause.

Syntax: Select column\_name1, …..,column\_namen from table name where condition;

Select command with order by clause:

Syntax: Select column\_name1, …..,column\_namen from table name where condition orderbycolmnname;

Select command to create a table:

Syntax: create table tablename as select \* from existing\_tablename;

Select command to insert records:

Syntax: insert into tablename( select columns from existing\_tablename);

**UPDATE** - It is used to alter the column values in a table. A single column may be updated or more than one column could be updated.

update<table name> set(fieldname-1 = value, fieldname-2 = value,…,fieldname-n = value)

[WHERE <condition/expression>];

**DELETE** - After inserting row in a table we can also delete them if required. The delete command consists of a from clause followed by an optional where clause.

delete from <table name> [where <condition/expression>];

**NESTED QUERIES:**

A subquery is a query within a query. In Oracle, you can create subqueries within your SQL statements. These subqueries can reside in the WHERE clause, the FROM clause, or the SELECT clause.

**JOINS:**

Join is a query in which data is returned from two or more tables.

How the join will be performed:

Step 1: Make the Cartesian product of the given tables.

Step 2: Check for the equality on common attributes for the given tables.

Natural join:

It returns the matching rows from the table that are being joined.

Syntax:

select<attribute> from TN where TN1.attribute=TN2.attribute.

Inner join:

The INNER JOIN selects all rows from both tables as long as there is a match found for the join predicate both tables.

Syntax:

select<attribute> from TN1 innerjoin TN2 on TN1.attribute=TN2.attribute.

Left outer join:

The LEFT OUTER JOIN returns all rows from the left table (table1), with the matching rows in the right table (table2). The result is NULL in the right side when there is no match.

Syntax:

select<attribute> from TN1 left outer join TN2 on TN1.attribute=TN2.attribute.

select<attribute> from TN where TN1.attribute(+)=TN2.attribute.

Right outer join:

The RIGHT OUTER JOIN returns all rows from the right table (table2), with the matching rows in the left table (table1). The result is NULL in the left side when there is no match.

Syntax:

select<attribute> from TN1 right outer join TN2 on TN1.attribute=TN2.attribute.

select<attribute> from TN where TN1.attribute=(+)TN2.attribute.

Full join:

The FULL OUTER JOIN returns all rows from the left table (table1) and from the right table (table2).

Syntax:

select<attribute> from TN1 full join TN2 on TN1.attribute=TN2.attribute.

**VIEWS:**

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.

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.

SQL CREATE VIEW Syntax

CREATE VIEW view\_name AS

SELECT column\_name(s)

FROM table\_name

WHERE condition

You can update a view by using the following syntax:

SQL CREATE OR REPLACE VIEW Syntax

CREATE OR REPLACE VIEW view\_name AS

SELECT column\_name(s)

FROM table\_name

WHERE condition

SQL Dropping a View

You can delete a view with the DROP VIEW command.

SQL DROP VIEW Syntax

DROP VIEW view\_name

**Aggregate Functions in SQL:**

Aggregate functions perform a calculation on a set of values and return a single value. Except for COUNT, aggregate functions ignore null values. Aggregate functions are frequently used with the GROUP BY clause of the SELECT statement.

Different Aggregate functions are as follows:

* AVG() - Returns the average value from the given table

Syntax: SELECT AVG(column\_name) FROM table\_name

* COUNT() - Returns the number of rows

Syntax:

SELECT COUNT(column\_name) FROM table\_name;

SELECT COUNT(\*) FROM table\_name;

SELECT COUNT(DISTINCT column\_name) FROM table\_name;

* FIRST() - Returns the first value

Syntax: SELECT FIRST(column\_name) FROM table\_name;

* LAST() - Returns the last value

Syntax: SELECT LAST(column\_name) FROM table\_name;

* MAX() - Returns the largest value

Syntax: SELECT MAX(column\_name) FROM table\_name;

* MIN() - Returns the smallest value

SELECT MIN(column\_name) FROM table\_name;

* SUM() - Returns the sum

SELECT SUM(column\_name) FROM table\_name;

**SQL Scalar functions:**

SQL scalar functions return a single value, based on the input value.

Different scalar functions are as follows:

* UCASE() - Converts a field to upper case

Syntax: SELECT UCASE(column\_name) FROM table\_name;

* LCASE() - Converts a field to lower case

Syntax: SELECT LCASE(column\_name) FROM table\_name;

* MID() - Extract middle characters from a text field

Syntax: SELECT MID(column\_name,start[,length]) AS some\_name FROM table\_name;

* LEN() - Returns the length of a text field

Syntax: SELECT LEN(column\_name) FROM table\_name;

* ROUND() - Rounds a numeric field to the number of decimals specified

Syntax: SELECT ROUND(column\_name,decimals) FROM table\_name;

* NOW() - Returns the current system date and time

Syntax: SELECT NOW() FROM table\_name;

* FORMAT() - Formats how a field is to be displayed

Syntax: SELECT FORMAT(column\_name,format) FROM table\_name;

**Implementation:**

**Results:**

**Experiment 6: Performing Query Operations using SQL statements**

**Objective:** Write and execute complex SQL query.

**Theory & Concepts:**

SQL is a database computer language designed for the retrieval and management of data in relational database. SQL stands for Structured Query Language. SQL is used to communicate with a database. It is the standard language for relational database management systems. SQL statements are used to perform tasks such as update data on a database, or retrieve data from a database. the standard SQL commands such as "Select", "Insert", "Update", "Delete", "Create", and "Drop" can be used to accomplish almost everything that one needs to do with a database.

**Implementation:**

**Results:**

**Experiment 7: Writing PL/SQL programs, procedures,triggers**

**Objective:** Write and execute Triggers and Procedures/ Functions using PL/ SQL in order to handle bulk data movement, data updates, and validations for data inputs.

**Theory & Concepts:**

**Triggers:**

A SQL trigger is a set of SQL statements stored in the database catalog. A SQL trigger is executed or fired whenever an event associated with a table occurs e.g., insert, update or delete.

Syntax

CREATE[OR REPLACE]TRIGGER[schema.]trigger

{BEFORE|AFTER|INSTEAD OF}

{DELETE|INSERT|UPDATE[OF column[,column].....]}

|OR {DELETE|INSERT|UPDATE[OF column,[,column]....]}]...

ON[schema.]{table|view}

REFERENCING(OLD[AS]old|NEW[AS]new...]

FOR EACH{ROW|STATEMENT}{WHEN(condition)}]

**Stored Procedures & Functions:**

A stored procedure is a segment of declarative SQL statements stored inside the database catalog. A stored procedure can be invoked by triggers, other stored procedures or applications such as Java, C#, PHP, etc.

Syntax:

CREATE

[DEFINER = { *user* | CURRENT\_USER }]

PROCEDURE *sp\_name* ([*proc\_parameter*[,...]])

[*characteristic* ...] *routine\_body*

CREATE

[DEFINER = { *user* | CURRENT\_USER }]

FUNCTION *sp\_name* ([*func\_parameter*[,...]])

RETURNS *type*

[*characteristic* ...] *routine\_body*

*proc\_parameter*:

[ IN | OUT | INOUT ] *param\_nametype*

*func\_parameter*:

*param\_nametype*

*type*:

*Any valid MySQL data type*

*characteristic*:

COMMENT '*string*'

| LANGUAGE SQL

| [NOT] DETERMINISTIC

| { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }

| SQL SECURITY { DEFINER | INVOKER }

*routine\_body*:

*Valid SQL routine statement*

**Implementation:**

**Results:**

**Experiment 8: Development of GUI for implementing the Case Study and perform operations through database connection string**

**Objective:** Create and perform Database operations using ODBC.

**Theory & Concepts:**

**ODBC** (**Open Database Connectivity**) is a standard programming language middleware API for accessing database management systems (DBMS). The designers of ODBC aimed to make it independent of database systems and operating systems; an application written using ODBC can be ported to other platforms, both on the client and server side, with few changes to the data access code.

ODBC accomplishes DBMS independence by using an **ODBC driver** as a translation layer between the application and the DBMS. The application uses ODBC functions through an **ODBC driver manager** with which it is linked, and the driver passes the query to the DBMS. An ODBC driver can be thought of as analogous to a printer or other driver, providing a standard set of functions for the application to use, and implementing DBMS-specific functionality.

e.g

Syntax for connecting to MySQL through java:

Class.forName ("com.mysql.jdbc.Driver").newInstance ();

conn = DriverManager.getConnection(url of database server,username,password);

**Implementation:**

**Results:**

**Experiment 9: Developing GUI for the Case Study & interacting with database**

**Objective:** Develop forms required to take data input from user. These forms should implement different input functionalities described in the problem statement.

**Theory & Concepts:**

Include the theory part related to the technology you have chosen for implementation.

**Implementation:**

//Include implementation details, flow of forms.

**Results:**

**Experiment 10: Generating Reports**

**Objective:** Generating information reports for data in database. This reports will be used for storing and presenting the information related to problem process and data flow.

**Theory & Concepts:**

Database reports are the formatted result of database queries and contain useful data for decision-making and analysis. Most good business applications contain a built-in reporting tool; this is simply a front-end interface that calls or runs back-end database queries that are formatted for easy application usage.

//Include theory regarding the reporting tool you are using

**Implementation:**

**Results:**