Group A
ASSIGNMENT NO.- 1

Title :

Identify the case study, formulate the problem statement and design ER /EER model.
[Identify entities,attributes realtionships,cardinalities,keys,generalization,specialization]

Requirement:

Operating System recommended :- 64-bit Open source Linux or its derivative
Programming tools recommended: - ERD plus, ER Win

THEORY:

ERDPlus is a web-based database modeling tool for creating entity relationship diagrams, relational schemas, star schemas, and SQL DDL statements. It automatically converts ER diagrams into relational schemas and works with most contemporary RDBMS tools.

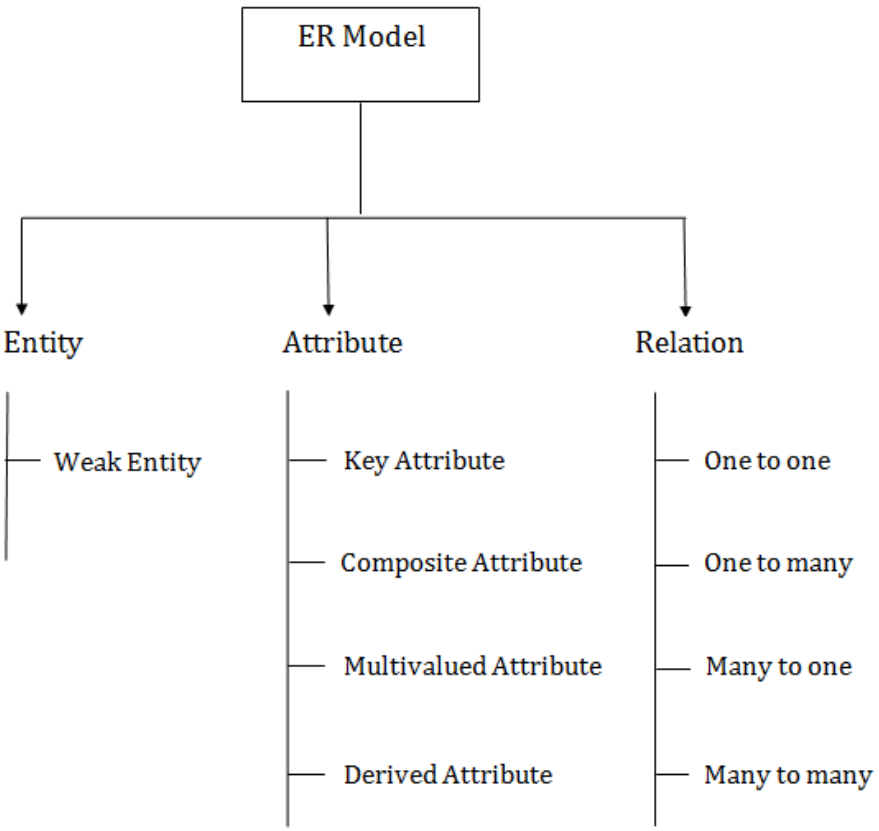
ER Model:

- ER model stands for an Entity-Relationship model. It is a high-level data model. This model is used to define the data elements and relationship for a specified system.
- It develops a conceptual design for the database. It also develops a very simple and easy to design view of data.
- In ER modeling, the database structure is portrayed as a diagram called an entity-relationship diagram

prepared by Prof.Kalyani Zore

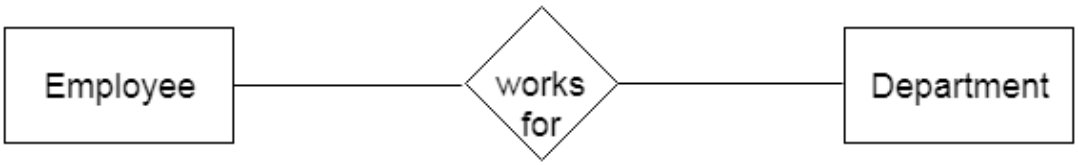
Component of ER Diagram:

- ER Model consists of mainly three components

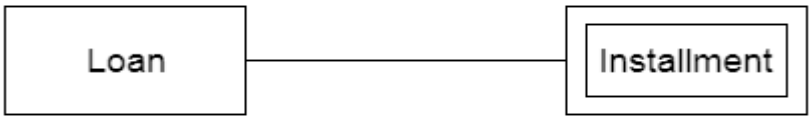


1.**Entity:**
An entity may be any object, class, person or place. In the ER diagram, an entity can be represented as rectangles.

Consider an organization as an example- manager, product, employee, department etc. can be taken as an entity.



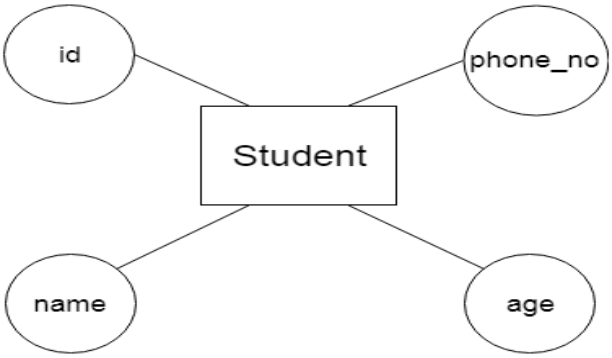
- **Weak Entity:** An entity that depends on another entity called a weak entity. The weak entity doesn't contain any key attribute of its own. The weak entity is represented by a double rectangle.



2.Attribute

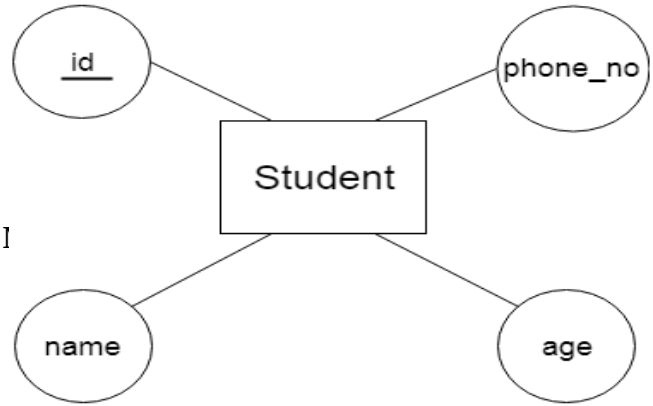
The attribute is used to describe the property of an entity. Eclipse is used to represent an attribute.
For example: id, age, contact number, name, etc. can be attributes of a student.

prepared by Prof.Kalyani Zore



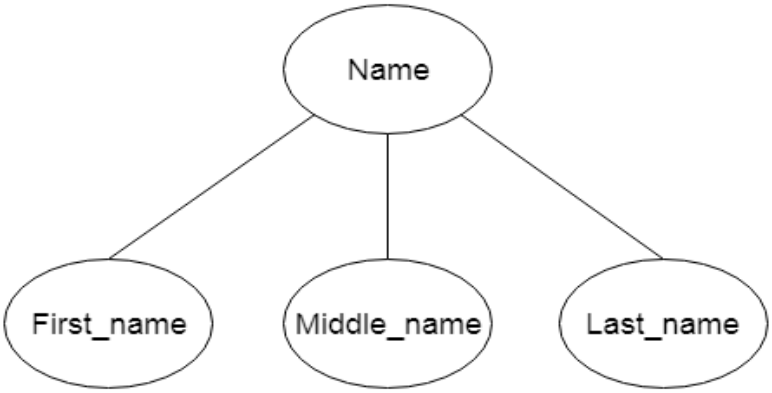
a. Key Attribute:

The key attribute is used to represent the main characteristics of an entity. It represents a primary key. The key attribute is represented by an ellipse with the text underlined.



b. Composite Attribute:

An attribute that composed of many other attributes is known as a composite attribute. The composite attribute is represented by an ellipse, and those ellipses are connected with an ellipse.

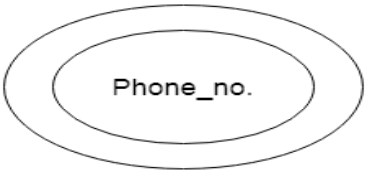


prepared by Prof.Kalyani Zore

c. Multivalued Attribute:

An attribute can have more than one value. These attributes are known as a multivalued attribute. The double oval is used to represent multivalued attribute.

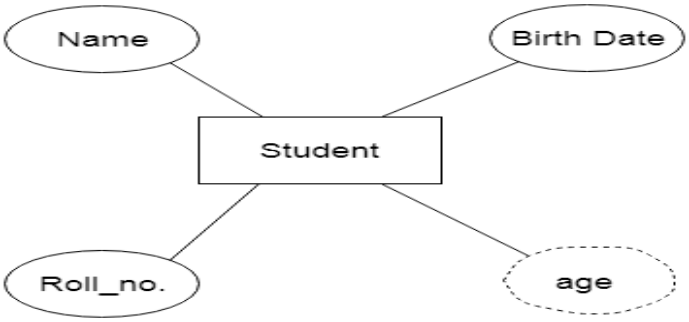
For example, a student can have more than one phone number.



d. Derived Attribute :

An attribute that can be derived from other attribute is known as a derived attribute. It can be represented by a dashed ellipse.

For example, A person's age changes over time and can be derived from another attribute like Date of birth.



3. Relationship:

A relationship is used to describe the relation between entities. Diamond or rhombus is used to represent the relationship.

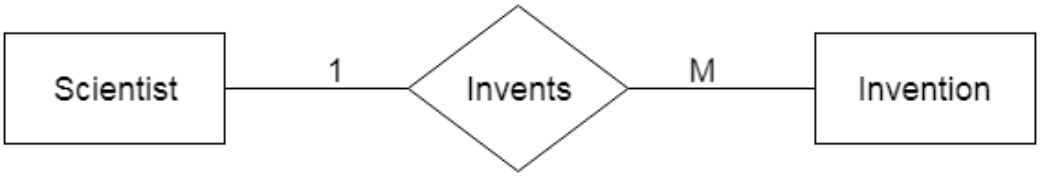
Types of relationship are as follows:

a. One-to-One Relationship:
When only one instance of an entity is associated with the relationship, then it is known as one to one relationship.

For example, A female can marry to one male, and a male can marry to one female.

b. One-to-many relationship:
When only one instance of the entity on the left, and more than one instance of an entity on the right associates with the relationship then this is known as a one-to-many relationship.

For example, Scientist can invent many inventions, but the invention is done by the only specific scientist.



c.Many-to-one relationship:

When more than one instance of the entity on the left, and only one instance of an entity on the right associates with the relationship then it is known as a many-to-one relationship.
For example, Student enrolls for only one course, but a course can have many students.

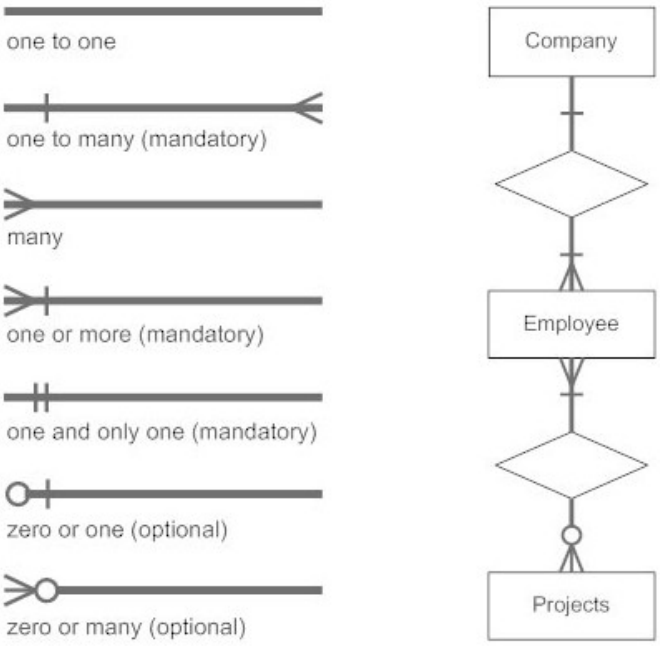
d. Many-to-many relations

hip:

When more than one instance of the entity on the left, and more than one instance of an entity on the right associates with the relationship then it is known as a many-to-many relationship.
For example, Employee can assign by many projects and project can have many employees.

Notation of ER diagram:

Database can be represented using the notations. In ER diagram, many notations are used to express the cardinality. These notations are as follows:



prepare i Zore

Fig: Notations of ER diagram

Case Study:
Consider following databases and draw ER diagram and convert entities and relationships to relation table for a given scenario.

1. COLLEGE DATABASE:

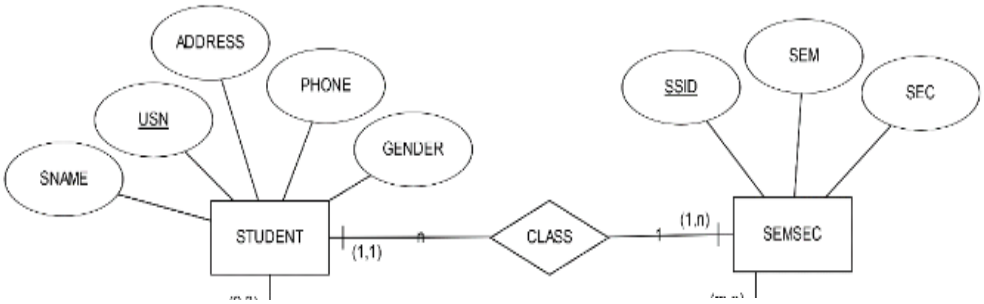
STUDENT (USN, SName, Address, Phone, Gender)
SEMSEC (SSID, Sem, Sec)
CLASS (USN, SSID)
SUBJECT (Subcode, Title, Sem, Credits)
IAMARKS (USN, Subcode, SSID, Test1, Test2, Test3, FinalIA)

2. COMPANY DATABASE:

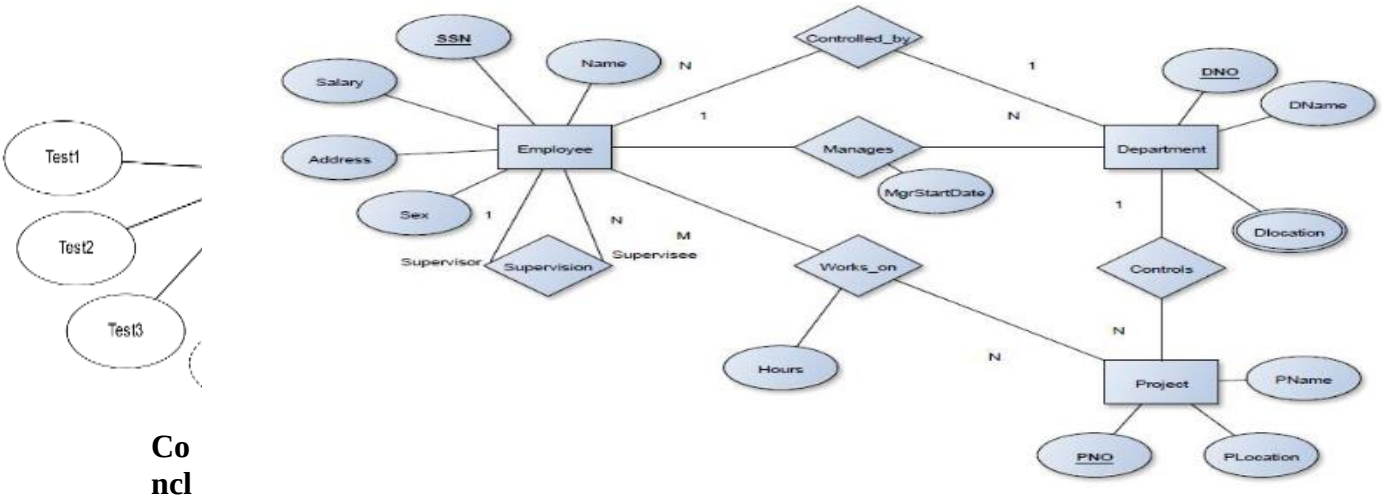
EMPLOYEE (SSN, Name, Address, Sex, Salary, SuperSSN, DNo)
DEPARTMENT (DNo, DName, MgrSSN, MgrStartDate)
DLOCATION (DNo,DLoc)
PROJECT (PNo, PName, PLocation, Dno) WORKS_ON (SSN, PNo, Hours)

College Database: E-R Diagram :

Mapping entities and relationships to relation table(Schema Diagram) :

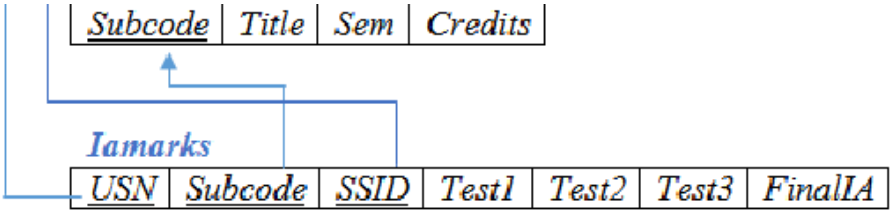


E-R Diagram



Conclusion:

In this assignment, we have studied concept of Entity model and notation used for drawing ER diagram.



Group A
ASSIGNMENT NO.- 2

AIM: Design and Develop SQL DDL statements which demonstrate the use of SQL objects such as Table, View, Index, Sequence, Synonym,different constraints etc.

prepared by Prof.Kalyani Zore

Requirement:

Hardware requirements: Any CPU with Pentium Processor or similar, 256 MB RAM or more,1 GB Hard Disk or more.

Software requirements: Ubuntu 14 Operating System, MySQL

THEORY:

a. Introduction to SQL:

SQL stands for Structured Query Language.

SQL lets you access and manipulate databases.

It is database programming language which along with programming language can be used to access data from database.

Commands of SQL are grouped into four languages.

1.**DDL :**

DDL is abbreviation of Data Definition Language. It is used to create and modify the structure of database objects in database.

Examples: CREATE, ALTER, DROP,RENAME,TRUNCATE statements

2.DML:

DML is abbreviation of Data Manipulation Language. It is used to retrieve, store, modify, delete, insert and update data in database.

Examples: SELECT, UPDATE, INSERT,DELETE statements

3.DCL:

DCL is abbreviation of Data Control Language. It is used to create roles, permissions, and referential integrity as well it is used to control access to database by securing it.

Examples: GRANT, REVOKE statements

4.TCL:

TCL is abbreviation of Transactional Control Language. It is used to manage different transactions occurring within a database

Examples: COMMIT, ROLLBACK statement

Data Definition Language (DDL):

1.Data definition Language (DDL) is used to create, rename, alter, modify, drop, replace, and delete tables, Indexes, Views .

2.The DDL part of SQL permits database tables to be created or deleted. It also define indexes (keys), specify links between tables, and impose constraints between tables. The most important DDL statements in SQL are:

- CREATE TABLE- Creates a new table
- ALTER TABLE- Modifies a table
- DROP TABLE- Deletes a table
- TRUNCATE -Use to truncate (delete all rows) a table.
- CREATE INDEX- Creates an index (search key)
- DROP INDEX- Deletes an index

prepared by Prof.Kalyani Zore

1. **The CREATE TABLE Statement** :

The CREATE TABLE statement is used to create a table in a database.

Syntax:

CREATE TABLE tablename
(attr1_name attr1_datatype(size) attr1_constraint,
attr2_name attr2_datatype(size) attr2_constraint,….)

SQL Constraints :

Constraints are used to limit the type of data that can go into a table.
Constraints can be specified when a table is created (with the CREATE TABLE statement) or after the table is created
(with the ALTER TABLE statement).

We will focus on the following constraints:

- NOT NULL
- UNIQUE
- PRIMARY KEY
- FOREIGN KEY

- CHECK
- DEFAULT

Add constraint after table creation using alter table option.

Syntax -: Alter table add constraint constraint_name constraint_type(Attr_name)
Example - Alter table stud add constraint prk1 primary key(rollno);
Drop constraint: Syntax- Drop Constraint Constraint_name;
Example - Drop constraint prk1;

2.**The Drop TABLE Statement** : Removes the table from the database.

Syntax: DROP TABLE table_name;

3. **The ALTER TABLE Statement :**
The ALTER TABLE statement is used to add, delete, or modify columns in an existing table.
Syntax : To add a column in a table, use the following syntax:

```
ALTER TABLE table_name
ADD column_name datatype;
```

To delete a column in a table, use the following syntax (notice that some database systems don't allow deleting a column):

```
ALTER TABLE table_name DROP COLUMN column_name;
```

To change the data type of a column in a table, use the following syntax:

```
ALTER TABLE table_name
MODIFY COLUMN column_name datatype;
```

prepared by Prof.Kalyani Zore

4. The RENAME TABLE Statement : Rename the old table to new table;
Syntax :

```
Rename old_tabname to new_tabname;
```

5. The TRUNCATE TABLE Statement : The ALTER TABLE Statement is used to truncate (delete all rows) a table.

Syntax :
To truncate a table, use following syntax : TRUNCATE TABLE table_name;

6. 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.

Syntax :

```
CREATE VIEW view_name AS
SELECT column_name(s)
FROM table_name
WHERE condition;
```

7. **SQL Dropping a View** : You can delete a view with the DROP VIEW command.

Syntax : DROP VIEW view_name;

8 . Create Index Statement :

1. Index in SQL is created on existing tables to retrieve the rows quickly. When there are thousands of records in a table, retrieving information will take a long time.
2. Therefore indexes are created on columns which are accessed frequently, so that the information can be retrieved quickly.
3. Indexes can be created on a single column or a group of columns. When a index is created, it first sorts the data and then it assigns a ROWID for each row.

Syntax : CREATE INDEX index_name
ON table_name (column_name1,column_name2...);
□□□ index_name is the name of the INDEX.
□□□ table_name is the name of the table to which the indexed column belongs.
column_name1, column_name2.. is the list of columns which make up the INDEX

9. Drop Index Statement :

Syntax : DROP INDEX index_name;

10. **Create Synonym statement :**

prepared by Prof Kalyani Zore

1. Use the CREATE SYNONYM statement to create a synonym, which is an alternative name for a table, view, sequence, procedure, stored function, package, materialized view.

2. Synonyms provide both data independence and location transparency. Synonyms permit applications to function without modification regardless of which user owns the table or view and regardless of which database holds the table or view.
3. You can refer to synonyms in the following DML statements: SELECT, INSERT, UPDATE, DELETE

Syntax :- Create synonym synonym-name for object-name;

Example-:- Create synonym synonym_name for table_name

Create synonym t for test

Conclusion: Studied the syntax for SQL DDL statements.

ASSIGNMENT NO.- 2(A)

AIM: Design at least 10 SQL queries for suitable database application using SQL DMLstatements: Insert, Select, Update, Delete with operators, functions, and set operator.

prepared by Prof.Kalyani Zore

Hardware requirements: Any CPU with Pentium Processor or similar, 256 MB RAM or more, 1 GB Hard Disk or more.

Software requirements:

Ubuntu 14 Operating System, MySQL

Theory:

DML : Data Manipulation Language (DML) statements are used for managing data in database. DML commands are not auto-committed. It means changes made by DML command are not permanent to database, it can be rolled back.

1) **INSERT command :**

Insert command is used to insert data into a table.
Following is its general syntax,

- INSERT into table-name values(data1,data2,..)

Lets see an example,
Consider a table **Student** with following fields.:
S_id S_Name age

INSERT into Student values(101,'Kiaan',10);
INSERT into Student values(102,'Kaveer,5);

The above command will insert a record into Student table.

S_id	S_Name	age
101	Kiaan	10
102	Kaveer	5

2) UPDATE command :

Update command is used to update a row of a table.

Following is its general syntax,

- UPDATE table-name set column-name = value where condition;

Lets see an example,

update Student set age=18 where s_id=102;

Example to Update multiple columns :

UPDATE Student set s_name='Nitya',age=11 where s_id=103;

3) **Delete command :** Delete command is used to delete data from a table.Delete command can also be used with the condition to delete a particular row.

prepared by Prof.Kalyani Zore

2023-24/Odd/v/CE/DMSL/LM

Following is its general syntax,

- DELETE from table-name;

Example to Delete all Records from a Table

DELETE from Student;

The above command will delete all the records from Student table.

Example to Delete a particular Record from a Table

Consider Student table

DELETE from Student where s_id=103;

SQL Functions : SQL provides many built-in functions to perform operations on data. These functions are useful while performing mathematical calculations, string concatenations, sub-strings etc.

SQL functions are divided into two catagories,

• Aggregrate Functions

• Scalar Functions

Aggregrate Functions :

These functions return a single value after calculating from a group of values.Following are some frequently used Aggregrate functions.

1) AVG()

Average returns average value after calculating from values in a numeric column.
Its general Syntax is,
SELECT AVG(column_name) from table_name
e.g. :SELECT avg(salary) from Emp;

2) COUNT()

Count returns the number of rows present in the table either based on some condition or without condition.
Its general Syntax is,
SELECT COUNT(column_name) from table-name;

Example using COUNT()

Consider following Emp table

eid	name	age	salary
401	Anu	22	9000
402	Shane	29	8000

prepared by Prof.Kalyani Zore

SQL query to count employees, satisfying specified condition is:
SELECT COUNT(name) from Emp where salary = 8000;
3) FIRST():

First function returns first value of a selected column
Syntax for FIRST function is,
SELECT FIRST(column_name) from table-name
SQL query:
SELECT FIRST(salary) from Emp;

4) LAST():

LAST return the return last value from selected column
Syntax of LAST function is,
SELECT LAST(column_name) from table-name

SQL query will be,
SELECT LAST(salary) from emp;

5) MAX()
MAX function returns maximum value from selected column of the table.

Syntax of MAX function is,
SELECT MAX(column_name) from table-name

SQL query to find Maximum salary is,
SELECT MAX(salary) from emp;

6) MIN()
MIN function returns minimum value from a selected column of the table.
Syntax for MIN function is,
SELECT MIN(column_name) from table-name

SQL query to find minimum salary is,
SELECT MIN(salary) from emp;

7) SUM()
SUM function returns total sum of a selected columns numeric values.
Syntax for SUM is,
SELECT SUM(column_name) from table-name

SQL query to find sum of salaries will be,
SELECT SUM(salary) from emp;

prepared by Prof.Kalyani Zore

Scalar Functions :
Scalar functions return a single value from an input value. Following are the frequently used Scalar Functions.

1) UCASE()
UCASE function is used to convert value of string column to Uppercase character.
Syntax of UCASE,
SELECT UCASE(column_name) from table-name
Example of UCASE()
SQL query for using UCASE is,
SELECT UCASE(name) from emp;

2) LCASE()
LCASE function is used to convert value of string column to Lowecase character.
Syntax for LCASE is:
SELECT LCASE(column_name) from table-name

3) MID()
MID function is used to extract substrings from column values of string type in a table.
Syntax for MID function is:
SELECT MID(column_name, start, length) from table-name

4) ROUND()
ROUND function is used to round a numeric field to number of nearest integer. It is used on Decimal point values. **Syntax** of Round function is,
SELECT ROUND(column_name, decimals) from table-name

Operators:

AND and OR operators are used with Where clause to make more precise conditions for fetching data from database by combining more than one condition together.

1) AND operator :
AND operator is used to set multiple conditions with Where clause.
Example of AND
SELECT * from Emp WHERE salary < 10000 AND age > 25

2) OR operator :
OR operator is also used to combine multiple conditions with Where clause. The only difference between AND and OR is their behaviour. When we use AND to combine two or more than two conditions, records satisfying all the condition will be in the result. But in case of OR, atleast one condition from the conditions specified must be satisfied by any record to be in the result.
Example of OR
SELECT * from Emp WHERE salary > 10000 OR age > 25

prepared by Prof.Kalyani Zore

Set Operation in SQL
SQL supports few Set operations to be performed on table data. These are used to get meaningful results from data, under different special conditions.

3) Union :
UNION is used to combine the results of two or more Select statements. However it will eliminate duplicate rows from its result set. In case of union, number of columns and datatype must be same in both the tables
Example of UNION :
select * from First
UNION select * from second

4) Union All :
This operation is similar to Union. But it also shows the duplicate rows.
Union All query will be like,
select * from First;
UNION ALL
select * from second
5) Intersect :

Intersect operation is used to combine two SELECT statements, but it only retuns the records which are common from both SELECT statements. In case of Intersect the number of columns and datatype must be same. MySQL does not support INTERSECT operator.

Intersect query will be,

select * from First
INTERSECT
select * from second

6) Minus:

Minus operation combines result of two Select statements and return only those result which belongs to first set of result. MySQL does not support Minus operator.

Minus query will be,

select * from First
MINUS select * from second

Conclusion: Studied the syntax for SQL DML statements.

Assignment No: 3
Group A
Assignment no.3

Aim: Design at least 10 SQL queries for suitable database application using SQL DML statements: all types of Join, Sub-Query and View.

prepared by Prof.Kalyani Zore

Hardware requirements:

Any CPU with Pentium Processor or similar, 256 MB RAM or more, 1 GB Hard Disk or more.

Software requirements: Ubuntu 14 Operating System, MySQL.

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:

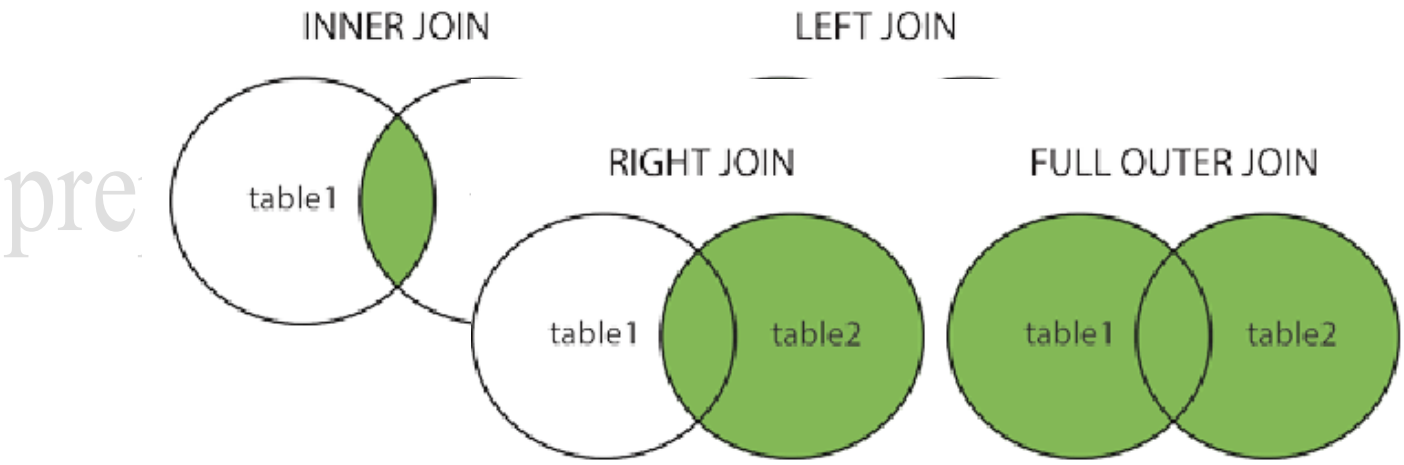
prepared by Prof.Kalyani Zore

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;
```

prepared by Prof.Kalyani Zore

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 SQL query nested inside a larger 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 :

```
Select    select_list
From      table
Where     expr operator
                ( Select    select_list
                  From      table );
```

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

prepared by Prof.Kalyani Zore

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

Conclusion:

Thus,we have successfully studied Joins, Subqueries and views and implemented SQL Queries.

prepared by Prof.Kalyani Zore

Group A
Assignment No: 4

Aim : Write a PL/SQL code block to calculate fine for a library book by accessing borrower information from the database.

Problem Statement:

Unnamed PL/SQL code block: Use of Control structure and Exception handling is mandatory. Write a PL/SQL block of code for the following requirements:-

Schema: 1. Borrower(Rollin, Name, DateofIssue, NameofBook, Status)
2. Fine(Roll_no,Date,Amt)

prepared by Prof Kalyani Zore

1) Accept roll_no & name of book from user.
2) Check the number of days (from date of issue), if days are between 15 to 30 then fine amount will be Rs 5per day.
3) If no. of days>30, per day fine will be Rs 50 per day & for days less than 30, Rs. 5 per day.
4) After submitting the book, status will change from I to R.
5) If condition of fine is true, then details will be stored into fine table

Hardware requirements: Any CPU with Pentium Processor or similar, 256 MB RAM or more, 1 GB Hard Disk or more.

Software requirements: Windows 7 Operating System, Oracle 11g, SQL developer

Theory: PL/SQL stands for Procedural Language extension of SQL. PL/SQL is a combination of SQL along with the procedural features of programming languages.It was developed by Oracle Corporation in the early 90's to enhance the capabilities of SQL.

Architecture of PL/SQL:

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.

- Anonymous Block
- Function
- *Library*
- Procedure
- Package Body
- Package Specification
- Trigger
- Type
- 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.

prepared by Prof.Kalyani Zore

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.

Basic Difference between SQL and PL/SQL :

In this section, we will discuss some differences between SQL and PL/SQL

SQL	PL/SQL

SQL is a single query that is used to perform DML and DDL operations.	1.PL/SQL is a block of codes that used to write the entire program blocks/ procedure/function, etc.
2.It is declarative, that defines what needs to be done, rather than how things need to bedone.	2.PL/SQL is procedural that defines how the things needs to be done.
3. Execute as a single statement.	3.Execute as a whole block.
4. Execute as a whole block.	4.Mainly used to create an application.
5. Interaction with Database server.	5.No interaction with the database server.
6. Cannot contain PL/SQL code in it.	6.It is an extension of SQL, so it can contain SQL inside it.

PL/SQL Block Structure:

Basic Syntax of PL/SQL which is a block-structured language; this means that the PL/SQL programs are divided and written in logical blocks of code. Each block consists of three sub-parts −

Sr.No	Sections & Description
1	**Declarations** This section starts with the keyword DECLARE. It is an optional section and defines all variables, cursors, subprograms, and other elements to be used in the program.
2	**Executable Commands** This section is enclosed between the keywords BEGIN and END and it is a mandatory section. It consists of the executable PL/SQL statements of the program. It should have at least one executable line of code, which may be just a NULL command to indicate that nothing should be executed.
3	**Exception Handling** This section starts with the keyword EXCEPTION. This optional section contains exception(s) that handle errors in the program.

prepared by Prof.Kalyani Zore

Every PL/SQL statement ends with a semicolon (;). PL/SQL blocks can be nested within other PL/SQL blocks using BEGIN and END. Following is the basic structure of a PL/SQL block −

```
DECLARE
<declarations section>

BEGIN
<executable command(s)>
```

```
EXCEPTION
<exception handling>

END;
```

The 'Hello World' Example :

```
DECLARE
message varchar2(20):= 'Hello, World!';
BEGIN
dbms_output.put_line(message);
END;
/
```

The end; line signals the end of the PL/SQL block.

PL/SQL Placeholders :

Placeholders are temporary storage area. PL/SQL Placeholders can be any of Variables, Constants and records. Oracle defines placeholders to store data temporarily, which are used to manipulate data during the execution of a PL SQL block.

Define PL/SQL Placeholders :

Depending on the kind of data you want to store, you can define placeholders with a name and a datatype. Few of the datatypes used to define placeholders are as given below.
Number (n,m) , Char (n) , Varchar2 (n) , Date , Long , Long raw, Raw, Blob, Clob, Nclob, Bfile.

prepared by Prof Kalyani Zore

PL/SQL Variables :

These are placeholders that store the values that can change through the PL/SQL Block.

General Syntax to declare a variable is

```
variable_name datatype [NOT NULL := value ];
```

variable_name is the name of the variable.
datatype is a valid PL/SQL datatype.
NOT NULL is an optional specification on the variable.
value or DEFAULT valueis also an optional specification, where you can initialize a variable. Each variable declaration is a separate statement and must be terminated by a semicolon.
For example, if you want to store the current salary of an employee, you can use a variable

```
DECLARE
    salary number (6);
```

The below example declares two variables, one of which is a not null.

```
DECLARE
        salary number(4);
      dept varchar2(10) NOT NULL := “HR Dept”;
```

The below example declares two variables, one of which is a not null.

```
DECLARE
        salary number(4);
       dept varchar2(10) NOT NULL := “HR Dept”;
```

The below example declares two variables, one of which is a not null.

```
DECLARE
        salary number(4);
        dept varchar2(10) NOT NULL := “HR Dept”;
```

prepared by Prof.Kalyani Zore

Conclusion:

Thus we have suucessfully implemented PL/SQL block to retrieve fine for issued library book by reading borrower information from the database.

Group A
Assignment No: 5

Aim: To Study and implement PL/SQL programming along with Procedures and Functions.

Problem Statement: Write a PL/SQL code block to study and implement stored procedure and function for displaying the result of student based on the grade obtained.

Hardware requirements:

- Any CPU with Pentium Processor or similar, 256 MB RAM or more, 1 GB Hard Disk or more.

prepared by Prof.Kalyani Zore

Software requirements: Ubuntu 14 Operating System, MySQL.

Theory:

PL/SQL (Procedural Language/Structured Query Language)
It is Oracle Corporation's proprietary procedural extension to the SQL database language, used in the Oracle database. Some other SQL database management systems offer similar extensions to the SQL language. PL/SQL's syntax strongly resembles that of Ada, and just like some Ada compilers of the 1980s, the PL/SQL runtime system uses Diana as intermediate representation.
The key strength of PL/SQL is its tight integration with the Oracle database.

Basic code structure in PL/SQL :

```
DECLARE
TYPE / item / FUNCTION / PROCEDURE declarations
BEGIN
Statements
EXCEPTION
EXCEPTION handlers
END;
```

The DECLARE and EXCEPTION sections are optional.

Simple Example:

```
DECLARE
number1 int;
number2 int:= 17; -- value default
text1 VARCHAR(12) := 'Hello world';
BEGIN
SELECT street_number
INTO number1
FROM address
WHERE name = 'Kiaan';
END;
```

FUNCTIONS :

Functions in PL/SQL are a collection of SQL and PL/SQL statements that perform a task and should return a value to the calling environment.

SYNTAX:

```
CREATE FUNCTION <function_name> [(input/output variable declarations)] RETURN
return_type
<IS|AS>
BEGIN
[declaration block]
<PL/SQL block WITH RETURN statement>
[EXCEPTION
EXCEPTION block]
END;
```

Prepared by Prof.Kalyani Zore

Example:

```
CREATE FUNCTION add(a in int,b out int,c in out int) return int IS
BEGIN
SELECT CONCAT ('a = ', a , ' b = ', b , ' c = ' , c);
SELECT CONCAT ('Addition Result = ');
return (a+b+c);
END;
```

PROCEDURES :

Procedures are the same as Functions, in that they are also used to perform some task with the difference being that procedures cannot be used in a SQL statement and although they can have multiple out parameters they do not return a value.

SYNTAX:

```
CREATE PROCEDURE <procedure_name> [(input/output variable declarations)]
```

```
BEGIN
[declaration block]
<PL/SQL block statements>
[EXCEPTION
EXCEPTION block]
END;
```

Example:
Create Procedure to check stock of items whose quantity is less than particular number and display result in temporary table 'stockcheck' and drop temp table after display.

```
CREATE PROCEDURE check_stock()
BEGIN
DECLARE SNO INT;
DECLARE ITEM CHAR(30);
DECLARE PRICE INT;
DECLARE TR INT;
DECLARE QA INT;
DECLARE C1 INT;
Declare C2 INT;
DECLARE CS cursor for select * from stock;
DECLARE CONTINUE HANDLER FOR NOT FOUND SET C1=1;
set C2=22;
create table stockcheck(stock_no int, item char(30), quantity_available int);
OPEN CS;
lable1: loop
FETCH CS INTO SNO,ITEM,PRICE,TR,QA;
if C1=1 then
leave lable1;
end if;
if QA <10 then
set C2=11;
insert into stockcheck values(SNO,ITEM,QA);
end if;
end loop;
if C2=22 then
Select "Enough stock for all items";
ELSE
select * from stockcheck;
drop table stcokcheck;
end if;
close CS;
END;//
```

Prepared by Prof.Kalyani Zore

Conclusion: *Thus successfully implemented procedures and function in PL/SQL*

Group A
Assignment No: 6

Aim: Write a PL/SQL block to create cursor to copy contents of one table into another. Avoid redundancy.

Problem Statement:
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_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.

prepared by Prof Kalyani Zore

Hardware requirements: Any CPU with Pentium Processor or similar, 256 MB RAM or more, 1 GB Hard Disk or more.

Software requirements: Ubuntu 14 Operating System, MySQL.

Theory:

Oracle creates a memory area, known as the context area, for processing an SQL statement, which contains all the information needed for processing the statement; for example, the number of rows processed, etc.

A cursor is a pointer to this context area. PL/SQL controls the context area through a cursor.
A cursor holds the rows (one or more) returned by a SQL statement. The set of rows the cursor holds is referred to as the active set.

You can name a cursor so that it could be referred to in a program to fetch and process the rows returned by the SQL statement, one at a time. There are two types of cursors −

- Implicit cursors
- Explicit cursors

Implicit Cursors :

Implicit cursors are automatically created by Oracle whenever an SQL statement is executed, when there is no explicit cursor for the statement. Programmers cannot control thei mplicit cursors and the information in it.

Whenever a DML statement (INSERT, UPDATE and DELETE) is issued, an implicit cursor is associated with this statement. For INSERT operations, the cursor holds the data that needs to be inserted. For UPDATE and DELETE operations, the cursor identifies the rows *that would be affected.*
In PL/SQL, you can refer to the most recent implicit cursor as the SQL cursor, which always has attributes such as %FOUND, %ISOPEN, %NOTFOUND, and %ROWCOUNT. The SQL cursor has additional attributes, %BULK_ROWCOUNT and %BULK_EXCEPTIONS designed for use with the FORALL statement. The following table provides the description of the most used attributes −

Sr no.	Attribute & Description
1.	%FOUND Returns TRUE if an INSERT, UPDATE, or DELETE statement affected one or more rows or a SELECT INTO statement returned one or more rows. Otherwise, it returns FALSE.
2.	%NOTFOUND The logical opposite of %FOUND. It returns TRUE if an INSERT, UPDATE, or DELETE statement affected no rows, or a SELECT INTO statement returned no rows. Otherwise, it returns FALSE.
3.	%ISOPEN Always returns FALSE for implicit cursors, because Oracle closes the SQL cursor automatically after executing its associated SQL statement.
4.	%ROWCOUNT Returns the number of rows affected by an INSERT, UPDATE, or DELETE statement, or returned by a SELECT INTO statement.

prepared by Prof.Kalyani Zore

Any SQL cursor attribute will be accessed as sql%attribute_name as shown below in the example.

Example :
We will be using the CUSTOMERS table we had created and used in the previous chapters.

```
Select  * from customers;

+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
+----+----------+-----+-----------+----------+
```

The following program will update the table and increase the salary of each customer by 500 and use the SQL%ROWCOUNT attribute to determine the number of rows affected −

```
DECLARE
  total_rows number(2);
BEGIN
  UPDATE customers
  SET salary = salary + 500;
  IF sql%notfound THEN
    dbms_output.put_line('no customers selected');
  ELSIF sql%found THEN
    total_rows := sql%rowcount;
    dbms_output.put_line( total_rows || ' customers selected ');
  END IF;
END;
/
```

When the above code is executed at the SQL prompt, it produces the following result −

```
6 customers selected

PL/SQL procedure successfully completed.
```

If you check the records in customers table, you will find that the rows have been updated −

```
Select * from customers;

+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2500.00 |
|  2 | Khilan   |  25 | Delhi     |  2000.00 |
|  3 | kaushik  |  23 | Kota      |  2500.00 |
|  4 | Chaitali |  25 | Mumbai    |  7000.00 |
|  5 | Hardik   |  27 | Bhopal    |  9000.00 |
|  6 | Komal    |  22 | MP        |  5000.00 |
+----+----------+-----+-----------+----------+
```

Explicit Cursors :

Explicit cursors are programmer-defined cursors for gaining more control over the context area. An explicit cursor should be defined in the declaration section of the PL/SQL Block. It is created on a SELECT Statement which returns more than one row.
The syntax for creating an explicit cursor is −

```
CURSOR cursor_name IS select_statement;
```

prepared by Prof.Kalyani Zore

Working with an explicit cursor includes the following steps –

- Declaring the cursor for initializing the memory
- Opening the cursor for allocating the memory
- Fetching the cursor for retrieving the data
- Closing the cursor to release the allocated memory

Declaring the Cursor :
Declaring the cursor defines the cursor with a name and the associated SELECT statement. For example −

```
CURSOR c_customers IS
   SELECT id, name, address FROM customers;
```

Opening the Cursor :

Opening the cursor allocates the memory for the cursor and makes it ready for fetching the rows returned by the SQL statement into it. For example, we will open the above defined cursor as follows −

```
OPEN c_customers;
```

Fetching the Cursor :
Fetching the cursor involves accessing one row at a time. For example, we will fetch rows from the above-opened cursor as follows −

```
FETCH c_customers INTO c_id, c_name, c_addr;
```

Closing the Cursor :
Closing the cursor means releasing the allocated memory. For example, we will close the above-opened cursor as follows −

```
CLOSE c_customers;
```

Example :

Following is a complete example to illustrate the concepts of explicit cursors &minua;

```
DECLARE
  c_id customers.id%type;
  c_name customerS.No.ame%type;
  c_addr customers.address%type;
  CURSOR c_customers is
    SELECT id, name, address FROM customers;
BEGIN
  OPEN c_customers;
  LOOP
  FETCH c_customers into c_id, c_name, c_addr;
    EXIT WHEN c_customers%notfound;
    dbms_output.put_line(c_id || ' ' || c_name || ' ' || c_addr);
  END LOOP;
  CLOSE c_customers;
END;
/
```

Conclusion: *Thus we have successfully implemented a PL/SQL block of code using parameterized Cursor.*

Group A
Assignment No: 7

Aim: Write a PL/SQL block to create trigger on Library table to keep track of updation and deletion of records.

Problem Statement:

Database Trigger (All Types: Row level and Statement level triggers, Before and After Triggers). Write a database trigger on Library table. The System should keep track of the records that are being updated or deleted. The old value of updated or deleted records should be added in Library_Audit table.

Hardware requirements: Any CPU with Pentium Processor or similar, 256 MB RAM or more, 1 GB Hard Disk or more.

Software requirements: Ubuntu 14 Operating System, MySQL.

Theory:

Triggers are stored programs, which are automatically executed or fired when some events occur. Triggers are, in fact, written to be executed in response to any of the following events −

prepared by Prof.Kalyani Zore

- A database manipulation (DML) statement (DELETE, INSERT, or UPDATE)
- A database definition (DDL) statement (CREATE, ALTER, or DROP).
- A database operation (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN).

Triggers can be defined on the table, view, schema, or database with which the event is associated.

Benefits of Triggers :
Triggers can be written for the following purposes −

- Generating derived column values automatically
- Enforcing referential integrity
- Event logging and storing information on table access
- Auditing
- Synchronous replication of tables
- Imposing security authorizations
- Preventing invalid transactions

Types of PL/SQL Triggers :

There are two types of triggers based on the which level it is triggered.

1) Row level trigger - An event is triggered for each row upated, inserted or deleted.
2) Statement level trigger - An event is triggered for each sql statement executed.

Creating Triggers :

The syntax for creating a trigger is −

```
CREATE [OR REPLACE ] TRIGGER trigger_name
{BEFORE | AFTER | INSTEAD OF }
{INSERT [OR] | UPDATE [OR] | DELETE}
[OF col_name]
ON table_name
[REFERENCING OLD AS o NEW AS n]
[FOR EACH ROW]
WHEN (condition)
DECLARE
   Declaration-statements
BEGIN
   Executable-statements
EXCEPTION
   Exception-handling-statements
END;
```

Where,

- CREATE [OR REPLACE] TRIGGER trigger_name − Creates or replaces an existing trigger with the trigger_name.
- {BEFORE | AFTER | INSTEAD OF} − This specifies when the trigger will be

executed. The INSTEAD OF clause is used for creating trigger on a view.

- {INSERT [OR] | UPDATE [OR] | DELETE} − This specifies the DML operation.
- [OF col_name] − This specifies the column name that will be updated.
- [ON table_name] − This specifies the name of the table associated with the trigger.
- [REFERENCING OLD AS o NEW AS n] − This allows you to refer new and old values for various DML statements, such as INSERT, UPDATE, and DELETE.
- [FOR EACH ROW] − This specifies a row-level trigger, i.e., the trigger will be executed for each row being affected. Otherwise the trigger will execute just once when the SQL statement is executed, which is called a table level trigger.
- WHEN (condition) − This provides a condition for rows for which the trigger would fire. This clause is valid only for row-level triggers.

Example:

To start with, we will be using the CUSTOMERS table we had created and used in the *chapters* −

prepared by Prof Kalyani Zore

```
Select * from customers;
```

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
+----+----------+-----+-----------+----------+
```

The following program creates a row-level trigger for the customers table that would fire for INSERT or UPDATE or DELETE operations performed on the CUSTOMERS table. This trigger will display the salary difference between the old values and new values −

```
CREATE OR REPLACE TRIGGER display_salary_changes

BEFORE DELETE OR INSERT OR UPDATE ON customers
```

```
FOR EACH ROW
WHEN (NEW.ID > 0)
DECLARE
   sal_diff number;
BEGIN
   sal_diff := :NEW.salary  - :OLD.salary;
   dbms_output.put_line('Old salary: ' || :OLD.salary);
   dbms_output.put_line('New salary: ' || :NEW.salary);
   dbms_output.put_line('Salary difference: ' || sal_diff);
END;
/
```

When the above code is executed at the SQL prompt, it produces the following result −

```
Trigger created.
```

Prepared by Prof. Kalyani Zore

he following points need to be considered here −
OLD and NEW references are not available for table-level triggers, rather you can use them for record-level triggers.

The above trigger has been written in such a way that it will fire before any DELETE or INSERT or UPDATE operation on the table, but you can write your trigger on a single or multiple operations, for example BEFORE DELETE, which will fire whenever a record will be deleted using the DELETE operation on the table.

Triggering a Trigger :

Let us perform some DML operations on the CUSTOMERS table. Here is one INSERT *statement, which will create a new record in the table* −

```
INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (7, 'Kriti', 22, 'HP', 7500.00 );
```

When a record is created in the CUSTOMERS table, the above create trigger, display_salary_changes will be fired and it will display result −

```
Old salary:
New salary: 7500
Salary difference:
```

Trigger for Library updation and deletion:

```
CREATE OR REPLACE TRIGGER BOOKS_AUDIT
BEFORE DELETE OR UPDATE ON library
REFERENCING OLD AS OLD NEW AS NEW
FOR EACH ROW
BEGIN
INSERT INTO library_audit
VALUES
( :old.id,
:old.book,
:old.author,
sysdate);
END;
```

prepared by Prof.Kalyani Zore

Conclusion: *Thus we have successfully implemented trigger to keep track of update and delete operation performed on library table.*

Group B
Assignment no.9

Aim : Develop MongoDB Queries using SAVE & Logical Operators.

Problem Statement:
Design and Develop MongoDB Queries using CRUD operations. (Use CRUD operations, SAVE method, logical operators)

Hardware requirements:
- Any CPU with Pentium Processor or similar, 256 MB
- RAM or more, 1 GB Hard Disk or more.

Software requirements: Ubuntu 14.04, MongoDB Packages.

Theory:
MongoDB is a free and open-source NoSQL document database used commonly in modern web applications. MongoDB works on concept of collection and document.

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.

prepared by Prof.Kalyani Zore

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.

CRUD Operations:
CRUD operations create, read, update, and delete documents.

These operations are considered to be the four basic functionalities of a repository .

CRUD operations can be mapped directly to database operations:

- Create matches insert
- Read matches select
- Update matches update
- Delete matches delete

Create Operations : Create or insert operations add new documents to a collection. If the collection does not currently exist, insert operations will create the collection.

db.collection.insert()
Syntax:
db.collection.insert(
<document or array of documents>,
{
writeConcern: <document>,
ordered: <boolean>
}
)

e.g.
1) db.products.insert({ item: "card", qty: 15 })
Will store record(document) as
{ "_id" : ObjectId("5063114bd386d8fadbd6b004"), "item" : "card", "qty" : 15 }

2) db.products.insert({ _id: 10, item: "box", qty: 20 })
Will store record (document) as
db.products.insert({ _id: 10, item: "box", qty: 20 })

Read Operations :

prepared by Prof.Kalyani Zore

Read operations retrieves documents from a collection; i.e. queries a collection for documents.

MongoDB provides the following methods to read documents from a collection:
 db.collection.find()

You can specify query filters or criteria that identify the documents to return.

Examples :

Find All Documents in a Collection :

The find() method with no parameters returns all documents from a collection and returns all fields for the documents. For example, the following operation returns all documents in the bios collection:

db.bios.find()

Find Documents that Match Query Criteria :

To find documents that match a set of selection criteria, call find() with the <criteria> parameter. The following operation returns all the documents from the collection products where qty is greater than 25:

db.products.find({ qty: { $gt: 25 } })

Update Operations :

MongoDB Update() Method:

The update() method updates the values in the existing document.

Syntax :

The basic syntax of update() method is as follows −

```
>db.COLLECTION_NAME.update(SELECTION_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})
```

Delete Operation:

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.

Syntax :
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'.

prepared by Prof.Kalyani Zore

```
>db.mycol.remove({'title':'MongoDB Overview'})
>db.mycol.find()
{ "_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}
{ "_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point Overview"}
>
```

Remove Only One :

If there are multiple records and you want to delete only the first record, then set justOne parameter in remove() method.

```
>db.COLLECTION_NAME.remove(DELETION_CRITERIA,1)
```

Remove All Documents :
If you don't specify deletion criteria, then MongoDB will delete whole documents from the collection. This is equivalent of SQL's truncate command.

```
>db.mycol.remove()
>db.mycol.find()
>
```

Conclusion: *Thus studied theMongoDB queries using CRUD operations.*

prepared by Prof.Kalyani Zore

Group B
Assignment No: 10

Aim : Aggregation and indexing with suitable example using MongoDB.

Requirements:
Software: - MongoDB.

Theory:
Aggregation :
Aggregations operations process data records and return computed results. Aggregation operations group values from multiple documents together, and can perform a variety of operations on the grouped data to return a single result. In sql count(*) and with group by is an equivalent of mongodb aggregation.

Syntax:

>db.COLLECTION_NAME.aggregate(AGGREGATE_OPERATION).

Aggregations are operations that process data records and return computed results. MongoDB provides a rich set of aggregation operations that examine and perform calculations on the data sets. Running data aggregation on the mongod instance simplifies application code and limits resource requirements.

prepared by Prof.Kalyani Zore

Like queries, aggregation operations in MongoDB use collections of documents as an input and return results in the form of one or more documents. The pipeline provides efficient data aggregation using native operations within MongoDB, and is the preferred method for data aggregation in MongoDB.

Aggregation Pipeline:

MongoDB's aggregation framework is modeled on the concept of data processing pipelines. Documents enter a multi-stage pipeline that transforms the documents into an aggregated result.

Indexing:

Indexes support the efficient execution of queries in MongoDB. Without indexes, MongoDB must scan every document in a collection to select those documents that match the query statement. These collection scans are inefficient because they require mongod to process a larger volume of data than an index for each operation.
Indexes are special data structures that store a small portion of the collection's data set in an easy to traverse form. The index stores the value of a specific field or set of fields, ordered by the value of the field.

Fundamentally, indexes in MongoDB are similar to indexes in other database systems.
MongoDB defines

indexes at the collection level and supports indexes on any field or sub-field of the documents in a MongoDB collection.
If an appropriate index exists for a query, MongoDB can use the index to limit the number of documents it must inspect. In some cases, MongoDB can use the data from the index to determine which documents match a query. The following diagram illustrates a query that selects documents using an index.

- Index Types

MongoDB provides a number of different index types to support specific types of data and queries.

- Default _id

All MongoDB collections have an index on the _id field that exists by default. If applications do not specify a value for _id the driver or the mongod will create an _id field with an ObjectId value.
The _id index is unique, and prevents clients from inserting two documents with the same value for the
_id field.

- Single Field

In addition to the MongoDB-defined _id index, MongoDB supports user-defined indexes on a single field of a document.

prepared by Prof.Kalyani Zore

- Compound Index

MongoDB also supports user-defined indexes on multiple fields. These compound indexes behave like single-field indexes; however, the query can select documents based on additional fields. The order of fields listed in a compound index has significance. For instance, if a compound index consists of { userid: 1, score: -1 }, the index sorts first by userid and then, within each userid value, sort by score.

- Multikey Index

MongoDB uses multikey indexes to index the content stored in arrays. If you index a field that holds an array value, MongoDB creates separate index entries for every element of the array.

These multikey indexes allow queries to select documents that contain arrays by matching on element or elements of the arrays. MongoDB automatically determines whether to create a multikey index if the indexed field contains an array value; you do not need to explicitly specify the multikey type.

- Geospatial Index

To support efficient queries of geospatial coordinate data, MongoDB provides two special indexes: 2d indexes that uses planar geometry when returning results and 2 sphere indexes that use spherical geometry to return results.

- Text Indexes

MongoDB provides a text index type that supports searching for string content in a collection.

These text indexes do not store language-specific stop words (e.g. "the", "a", "or") and stem the

words in a collection to only store root words.

- Hashed Indexes

To support hash based sharding, MongoDB provides a hashed index type, which indexes the hash of the value of a field. These indexes have a more random distribution of values along their range, but only support equality matches and cannot support range-based queries.

Index Properties:

- **Sparse Indexes:**

The sparse property of an index ensures that the index only contain entries for documents that have the indexed field. The index skips documents that do not have the indexed field.

- Unique Indexes

The unique property for an index causes MongoDB to reject duplicate values for the indexed field.

Create Indexes to Support single Queries :

An index supports a query when the index contains all the fields scanned by the query. The query scans the index and not the collection. Creating indexes that support queries results in greatly increased query performance.

This document describes strategies for creating indexes that support queries.

prepared by Prof.Kalyani Zore

Create a Single-Key Index if All Queries Use the Same, Single Key :

If you only ever query on a single key in a given collection, then you need to create just one single-key index for that collection. For example, you might create an index on category in the product collection:

```
db.products.ensureIndex( { "category": 1 } )
```

This allows you both options. You can query on just category, and you also can query on category combined with item. A single compound index on multiple fields can support all the queries that search a "prefix" subset of those fields.

Example
The following index on a collection:
{ x: 1, y: 1, z: 1 }

Can support queries that the following indexes support:
{ x: 1 }
{ x: 1, y: 1 }

There are some situations where the prefix indexes may offer better query performance: for example if z is a large array.
The { x: 1, y: 1, z: 1 } index can also support many of the same queries as the following index:
{ x: 1, z: 1 }

Also, { x: 1, z: 1 } has an additional use. Given the following query:
db.collection.find({ x: 5 }).sort({ z: 1})

The { x: 1, z: 1 } index supports both the query and the sort operation, while the { x: 1, y: 1, z:1 } index only supports the query.

Create Compound Indexes to Support Several Different Queries :

If you sometimes query on only one key and at other times query on that key combined with a second key, then creating a compound index is more efficient than creating a single-key index. MongoDB will use the compound index for both queries. For example, you might create an index on both category and item.

db.products.ensureIndex({ "category": 1, "item": 1 })

Example :

The following index on a collection:
{ x: 1, y: 1, z: 1 }
Can support queries that the following indexes support:
{ x: 1 }
{ x: 1, y: 1 }
There are some situations where the prefix indexes may offer better query performance: for example if z is a large array.
The { x: 1, y: 1, z: 1 } index can also support many of the same queries as the following index:
{ x: 1, z: 1 }
Also, { x: 1, z: 1 } has an additional use.
Given the following query:
db.collection.find({ x: 5 }).sort({ z: 1})

prepared by Prof.Kalyani Zore

Use Indexes to Sort Query Results :

In MongoDB, sort operations can obtain the sort order by retrieving documents based on the ordering in an index. If the query planner cannot obtain the sort order from an index, it will sort the results in memory. Sort operations that use an index often have better performance than those that do not use an index. In addition, sort operations that do not use an index will abort when *they use 32 megabytes of memory.*

Sort with a Single Field Index :

If an ascending or a descending index is on a single field, the sort operation on the field can be in either direction.
For example, create an ascending index on the field a for a collection records:
db.records.ensureIndex({ a: 1 })
This index can support an ascending sort on a:
db.records.find().sort({ a: 1 })
The index can also support the following descending sort on a by traversing the index in reverse order:

db.records.find().sort({ a: -1 }).

Create Indexes that Support Covered Queries :

A covered query is a query in which:

- *all the fields in the query are part of an index, and*
- all the fields returned in the results are in the same index.

Because the index "covers" the query, MongoDB can both match the query conditions and return the results using only the index; MongoDB does not need to look at the documents, only the index, to fulfill the query.

Querying only the index can be much faster than querying documents outside of the index. Index keys are typically smaller than the documents they catalog, and indexes are typically available in RAM or located sequentially on disk.

MongoDB automatically uses an index that covers a query when possible. To ensure that an index can cover a query, create an index that includes all the fields listed in the query document and in the query result. You can specify the fields to return in the query results with a projection document. By default, MongoDB includes the _id field in the query result. So, if the index does not include the _id field, then you must exclude the _id field (i.e. _id: 0) from the query results.

Example :

prepared by Prof.Kalyani Zore

Given collection users with an index on the fields user and status, as created by the following option:

db.users.ensureIndex({ status: 1, user: 1 })

Then, this index will cover the following query which selects on the status field and returns only the user field:

db.users.find({ status: "A" }, { user: 1, _id: 0 })

In the operation, the projection document explicitly specifies _id: 0 to exclude the _id field from the result since the index is only on the status and the user fields.

If the projection document does not specify the exclusion of the _id field, the query returns the _id field. The following query is not covered by the index on the status and the user fields because with the projection document { user: 1 }, the query returns both the user field and the _id field:

db.users.find({ status: "A" }, { user: 1 })

Conclusion:

Group B
Assignment no.12

Aim: Write a program to implement MongoDB database connectivity with PHP.

Hardware requirements: Any CPU with Pentium Processor or similar, 256 MB
 RAM or more, 1 GB Hard Disk or more.

Software requirements: Ubuntu 14.04 or Windows 7, PHP5.2, Any Javascript enabled browser.

Theory:
To use MongoDB with PHP, we need to install additional MongoDB PHP driver.

Make a Connection and Select a Database :
To make a connection, you need to specify the database name, if the database doesn't exist then MongoDB creates it automatically.

Following is the code snippet to connect to the database −

```
 <?php
 // connect to mongodb
$m = new MongoClient();
echo "Connection to database successfully";
// select a database
$db = $m->mydb;
echo "Database mydb selected";
?>
```

prepared by Prof.Kalyani Zore

When the program is executed, it will produce the following result −

```
Connection to database
successfully Database mydb
selected
```

Create a Collection
Following is the code snippet to create a collection −

```
<?php
// connect to mongodb
$m = new MongoClient();
echo "Connection to database successfully";
// select a database
$db = $m->mydb;
echo "Database mydb selected";
$collection = $db-
>createCollection("mycol"); echo
```

"Collection created succsessfully";
?>
When the program is executed, it will produce the following result −

Connection to database
successfully Database mydb
selected
Collection created succsessfully

Insert a Document :

To insert a document into MongoDB, insert() method is used.

Following is the code snippet to insert a document −

```
<?php
// connect to mongodb
$m = new MongoClient();
echo "Connection to database successfully";
// select a database
$db = $m->mydb;

echo "Database mydb selected";
$collection = $db->mycol;
echo "Collection selected succsessfully";
$document = array(
"title" =>
"MongoDB",
"description" =>
"database", "likes" => 100,
"url" =>
"http://www.mongo.com/mongodb/", "by",
"JIT"
);
```

Prepared by Prof.Kalyani Zore

When the program is executed, it will produce the following result −
Connection to database
successfully Database mydb
selected
Collection selected
succsessfully Document
inserted successfully

Find All Documents :

To select all documents from the collection, find() method is used.

Following is the code snippet to select all documents −

```
<?php
// connect to mongodb
$m = new MongoClient();
echo "Connection to database successfully";
// select a database
$db = $m->mydb;
echo "Database mydb selected";
$collection = $db->mycol;
echo "Collection selected succsessfully";
$cursor = $collection->find();
// iterate cursor to display title of documents
foreach ($cursor as $document)
{ echo $document["title"] . "\
n";
}
?>
```

When the program is executed, it will produce the following result −

```
Connection to database
successfully Database mydb
selected
Collection selected succsessfully
```

Update a Document :

prepared by Prof.Kalyani Zore

To update a document, you need to use the update() method.
In the following example, we will update the title of inserted document to MongoDB. Following is the code snippet to update a document −

```
<?php
// connect to mongodb
$m = new MongoClient();
echo "Connection to database successfully";
// select a database
$db = $m->mydb;
echo "Database mydb selected";
$collection = $db->mycol;
echo "Collection selected succsessfully";
// now update the document
$collection->update(array("title"=>"MongoDB"),
array('$set'=>array("title"=>"MongoDBNew")));

echo "Document updated successfully";
// now display the updated document
$cursor = $collection->find();
// iterate cursor to display title of
documents echo "Updated document";
foreach ($cursor as $document)
```

```
{ echo $document["title"] . "\
n";
}
?>
```

When the program is executed, it will produce the following result −

```
Connection to database successfully
Database mydb selected
Collection selected succsessfully
Document updated successfully
Updated document {
"title": "MongoDBNew"
}
```

Delete a Document :

To delete a document, you need to use remove() method.
In the following example, we will remove the documents that has the title MongoDB

Tutorial. Following is the code snippet to delete a document −

```
<?php
// connect to mongodb
$m = new MongoClient();
echo "Connection to database successfully";
// select a database
$db = $m->mydb;
echo "Database mydb selected";
$collection = $db->mycol;
echo "Collection selected succsessfully";

// now remove the document
$collection->remove(array("title"=>"MongoDBNew"),false);
echo "Documents deleted successfully";
// now display the available documents
$cursor = $collection->find();
// iterate cursor to display title of
documents echo "Updated document";
foreach ($cursor as $document)
{ echo $document["title"] . "\
n";
}
?>
```

When the program is executed, it will produce the following result −

```
Connection to database successfully
Database mydbselected
Collection selected succsessfully
Documents deleted successfully
```

In the above example, the second parameter is boolean type and used for

Prepared by Prof.Kalyani Zore

justOne field of remove() method.

Conclusion: Thus we learnt and successfully implemented MongoDB database connectivity with MongoDB.

prepared by Prof.Kalyani Zore