**Crud Dr. D. Y. Patil Pratishthan’s**

**DR. D. Y. PATIL INSTITUTE OF ENGINEERING, MANAGEMENT & RESEARCH**

**Approved by A.I.C.T.E, New Delhi , Maharashtra State Government, Affiliated to Savitribai Phule Pune University**

Sector No. 29, PCNTDA , Nigidi Pradhikaran, Akurdi, Pune 411044. Phone: 020–27654470, Fax: 020-27656566  
Website : [www.dypiemr.ac.in](http://www.dypiemr.ac.in) Email : [principal.dypiemr@gmail.com](mailto:principal.dypiemr@gmail.com)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**DEPARTMENT**

**OF**

**COMPUTER ENGINEERING**

**LAB MANUAL**

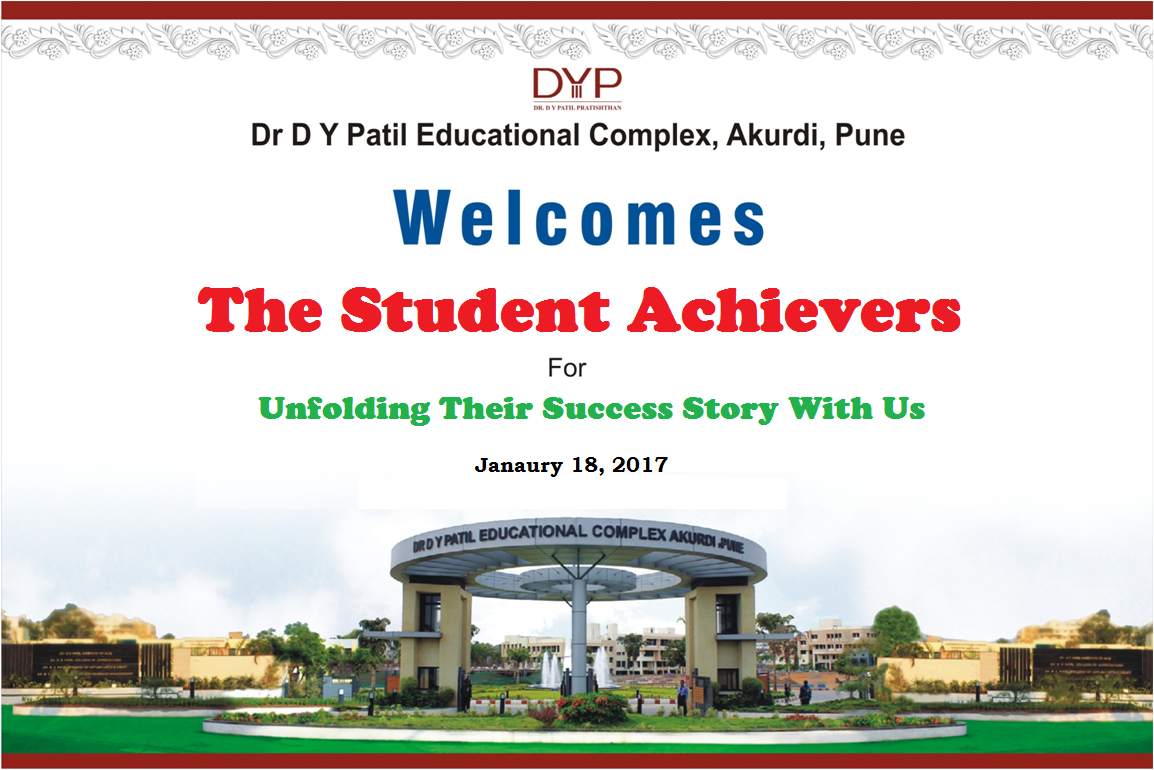
**Database Management System Laboratory**

**(Third Year Engineering)**

**Semester – I**

**Prepared by :** Mrs. Nalini Yadav

Mrs. Tanuja Lonhari



# Table of Contents

|  |  |  |  |
| --- | --- | --- | --- |
| Sr.No |  | Title of the Experiment | Page No |
| **Group A** | | | |
| 1 | A1 | Study of open source relational database: MYSQL. | 3-7 |
| 2 | A2 | Design and develop SQL DDL statements which demonstrates use of SQL objects such as Table, View, Index, Sequence, and Synonym. | 8-12 |
| 3 | A3 | Design at least 10 subqueries for suitable database application using DML statements: Insert, select, Update and Delete with operators, functions and set operators. | 13-17 |
| 4 | A4 | Design at least 10 subqueries for suitable database application using DML statements: all types of joins, subquery and Views. | 18-21 |
| 5 | A5 | Write PL/SQL code block: use of control structures and exception handling. | 22-24 |
| 6 | A6 | Write a PL/SQL block of code using parameterized cursor that will merge the data available in the newly created table. | 25-31 |
| 7 | A7 | Write PL/SQL stored procedure and stored function. | 32-37 |
| 8 | A8 | Write a database trigger on a 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. | 38-39 |
| **Group B** | | | |
| 9 | B1 | Study on open source NOSQL database: MongoDB. | 40-44 |
| 10 | B2 | Design and develop MongoDB queries using CRUD operations, SAVE method and logical operators. | 45-49 |
| 11 | B3 | Implement Indexing and Aggregation using MongoDB. | 50-56 |
| 12 | B4 | Implement Map-reduce operation with suitable using MongoDB. | 57-60 |
| 13 | B5 | Design and implement any 5 queries using MongoDB. | 61-66 |
| **Group C** | | | |
| 14 | C1 | Write a program to implement MongoDB database connectivity with PHP/PYTHON/JAVA implement database navigation operations using JDBC/ODBC. | 67-70 |
| 15 | C2 | Implement MYSQL/ORACLE database connectivity with PHP/PYTHON/JAVA implement database navigation operations using JDBC/ODBC. | 71-81 |

**Assignment No. A1**

* **Title:** Study of Open Source Relational Databases : MySQL
* **Objective:** Study and understand Open Source Relational Databases concept : MySQL
* **Theory:**

SQL is a standard language for storing, manipulating and retrieving data in databases.

**What is SQL?**

* SQL stands for Structured Query Language
* SQL lets you access and manipulate databases
* SQL is an ANSI (American National Standards Institute) standard

**What Can SQL do?**

* SQL can execute queries against a database
* SQL can retrieve data from a database
* SQL can insert records in a database
* SQL can update records in a database
* SQL can delete records from a database
* SQL can create new databases
* SQL can create new tables in a database
* SQL can create stored procedures in a database
* SQL can create views in a database
* SQL can set permissions on tables, procedures, and views

## SQL is a Standard - BUT....

Although SQL is an ANSI (American National Standards Institute) standard, there are different versions of the SQL language.

However, to be compliant with the ANSI standard, they all support at least the major commands (such as SELECT, UPDATE, DELETE, INSERT, WHERE) in a similar manner.

**Note:** Most of the SQL database programs also have their own proprietary extensions in addition to the SQL standard!

**Using SQL in Your Web Site**

To build a web site that shows data from a database, you will need:

* An RDBMS database program (i.e. MS Access, SQL Server, MySQL)
* To use a server-side scripting language, like PHP or ASP
* To use SQL to get the data you want
* To use HTML / CSS to style the page

## **Features**

MySQL is offered under two different editions: the open source MySQL Community Server and the proprietary Enterprise Server.MySQL Enterprise Server is differentiated by a series of proprietary extensions which install as server plugins, but otherwise shares the version numbering system and is built from the same code base.

Major features as available in MySQL 5.6:

* A broad subset of ANSI SQL 99, as well as extensions
* Cross-platform support
* Stored procedures, using a procedural language that closely adheres to SQL/PSM
* Triggers
* Cursors
* Updatable views
* Online DDL when using the InnoDB Storage Engine.
* Information schema
* Performance Schema that collects and aggregates statistics about server execution and query performance for monitoring purposes.
* A set of SQL Mode options to control runtime behavior, including a strict mode to better adhere to SQL standards.
* X/Open XA distributed transaction processing (DTP) support; two phase commit as part of this, using the default InnoDB storage engine
* Transactions with savepoints when using the default InnoDB Storage Engine. The NDB Cluster Storage Engine also supports transactions.
* ACID compliance when using InnoDB and NDB Cluster Storage Engines
* SSL support
* Query caching
* Sub-SELECTs (i.e. nested SELECTs)
* Built-in replication support (i.e., master-master replication and master-slave replication) with one master per slave, many slaves per master. Multi-master replication is provided in MySQL Cluster, and multi-master support can be added to unclustered configurations using Galera Cluster.
* Full-text indexing and searching
* Embedded database library
* Unicode support
* Partitioned tables with pruning of partitions in optimizer
* Shared-nothing clustering through MySQL Cluster
* Multiple storage engines, allowing one to choose the one that is most effective for each table in the application.
* Native storage engines InnoDB, MyISAM, Merge, Memory (heap), Federated, Archive, CSV, Blackhole, NDB Cluster.
* Commit grouping, gathering multiple transactions from multiple connections together to increase the number of commits per second.

## RDBMS

* RDBMS stands for Relational Database Management System.
* RDBMS is the basis for SQL, and for all modern database systems such as MS SQL Server, IBM DB2, Oracle, MySQL, and Microsoft Access.
* The data in RDBMS is stored in database objects called tables. A table is a collection of related data entries and it consists of columns and rows.

Look at the "Customers" table:

Every table is broken up into smaller entities called fields. The fields in the Customers table consist of CustomerID, CustomerName, ContactName, Address, City and PostalCode. A field is a column in a table that is designed to maintain specific information about every record in the table.

A **record**, also called a **row**, is each individual entry that exists in a table. For example, there are 91 records in the above Customers table. A record is a horizontal entity in a table.

A column is a vertical entity in a table that contains all information associated with a specific field in a table.

## Database Tables

A database most often contains one or more tables. Each table is identified by a name (e.g. "Customers" or "Orders"). Tables contain records (rows) with data.

In this tutorial we will use the well-known Northwind sample database (included in MS Access and MS SQL Server).

Below is a selection from the "Customers" table:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 1 | Alfreds Futterkiste | Maria Anders | Obere Str. 57 | Berlin | 12209 | Germany |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Avda. de la Constitución 2222 | México D.F. | 05021 | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mataderos 2312 | México D.F. | 05023 | Mexico |
| 4 | Around the Horn | Thomas Hardy | 120 Hanover Sq. | London | WA1 1DP | UK |
| 5 | Berglunds snabbköp | Christina Berglund | Berguvsvägen 8 | Luleå | S-958 22 | Sweden |

The table above contains five records (one for each customer) and seven columns (CustomerID, CustomerName, ContactName, Address, City, PostalCode, and Country).

## SQL Statements

Most of the actions you need to perform on a database are done with SQL statements.

The following SQL statement selects all the records in the "Customers" table:

### Example

SELECT \* FROM Customers;

* **SQL keywords are NOT case sensitive: select is the same as SELECT**

Semicolon after SQL Statements?

Some database systems require a semicolon at the end of each SQL statement.

Semicolon is the standard way to separate each SQL statement in database systems that allow more than one SQL statement to be executed in the same call to the server.

**Some of The Most Important SQL Commands**

* **SELECT** - extracts data from a database
* **UPDATE** - updates data in a database
* **DELETE** - deletes data from a database
* **INSERT INTO** - inserts new data into a database
* **CREATE DATABASE** - creates a new database
* **ALTER DATABASE** - modifies a database
* **CREATE TABLE** - creates a new table
* **ALTER TABLE** - modifies a table
* **DROP TABLE** - deletes a table
* **CREATE INDEX** - creates an index (search key)
* **DROP INDEX** - deletes an index

**Conclusion :**

We have successfully Study and understand the basic theoretical concepts of Open Source Relational Databases MySQL

**Assignment No. A2**

* **Title:** Design and Develop SQL DDL statements which demonstrate the use of SQL objects such as Table, View, Index, Sequence, Synonym
* **Objective:** To implement Table, View, Index, Sequence, Synonym
* **Problem Statement:** Write a SQL DDL statements which demonstrate the use of SQL objects such as Table, View, Index, Sequence, Synonym
* **Hardware and Software requirement:**

1. 64-bit Dell Machine
2. Linux OS
3. MySQL

* **Theory:**

To begin with, the table creation command requires the following details

* Name of the table
* Name of the fields
* Definitions for each field

### Syntax

Here is a generic SQL syntax to create a MySQL table −

CREATE TABLE table\_name (column\_name column\_type);

Now, we will create the following table in the **TUTORIALS** database.

tutorials\_tbl(

tutorial\_id INT NOT NULL AUTO\_INCREMENT,

tutorial\_title VARCHAR(100) NOT NULL,

tutorial\_author VARCHAR(40) NOT NULL,

submission\_date DATE,

PRIMARY KEY ( tutorial\_id )

);

Here, a few items need explanation −

* Field Attribute **NOT NULL** is being used because we do not want this field to be NULL. So, if a user will try to create a record with a NULL value, then MySQL will raise an error.
* Field Attribute **AUTO\_INCREMENT** tells MySQL to go ahead and add the next available number to the id field.
* Keyword **PRIMARY KEY** is used to define a column as a primary key. You can use multiple columns separated by a comma to define a primary key.

# Drop MySQL Tables

It is very easy to drop an existing MySQL table, but you need to be very careful while deleting any existing table because the data lost will not be recovered after deleting a table.

### Syntax

Here is a generic SQL syntax to drop a MySQL table −

DROP TABLE table\_name ;

## **Dropping Tables from the Command Prompt**

To drop tables from the command prompt, we need to execute the DROP TABLE SQL command at the mysql> prompt.

### Example

The following program is an example which deletes the **tutorials\_tbl** −

root@host# mysql -u root -p

Enter password:\*\*\*\*\*\*\*

mysql> use TUTORIALS;

Database changed

mysql> DROP TABLE tutorials\_tbl

Query OK, 0 rows affected (0.8 sec)

mysql>

**MySQL - INDEXES**

* A database index is a data structure that improves the speed of operations in a table. Indexes can be created using one or more columns, providing the basis for both rapid random lookups and efficient ordering of access to records.
* While creating index, it should be taken into consideration which all columns will be used to make SQL queries and create one or more indexes on those columns.
* Practically, indexes are also a type of tables, which keep primary key or index field and a pointer to each record into the actual table.
* The users cannot see the indexes, they are just used to speed up queries and will be used by the Database Search Engine to locate records very fast.
* The INSERT and UPDATE statements take more time on tables having indexes, whereas the SELECT statements become fast on those tables. The reason is that while doing insert or update, a database needs to insert or update the index values as well.

## **Simple and Unique Index**

You can create a unique index on a table. A unique index means that two rows cannot have the same index value. Here is the syntax to create an Index on a table.

CREATE UNIQUE INDEX index\_name

ON table\_name ( column1, column2,...);

You can use one or more columns to create an index.

For example, we can create an index on **tutorials\_tbl** using **tutorial\_author**.

CREATE UNIQUE INDEX AUTHOR\_INDEX

ON tutorials\_tbl (tutorial\_author)

You can create a simple index on a table. Just omit the **UNIQUE** keyword from the query to create a simple index. A Simple index allows duplicate values in a table.

If you want to index the values in a column in a descending order, you can add the reserved word DESC after the column name.

mysql> CREATE UNIQUE INDEX AUTHOR\_INDEX

ON tutorials\_tbl (tutorial\_author DESC)

## **ALTER command to add and drop INDEX**

There are four types of statements for adding indexes to a table −

* **ALTER TABLE tbl\_name ADD PRIMARY KEY (column\_list)** − This statement adds a **PRIMARY KEY**, which means that the indexed values must be unique and cannot be NULL.
* **ALTER TABLE tbl\_name ADD UNIQUE index\_name (column\_list)** − This statement creates an index for which the values must be unique (except for the NULL values, which may appear multiple times).
* **ALTER TABLE tbl\_name ADD INDEX index\_name (column\_list)**− This adds an ordinary index in which any value may appear more than once.
* **ALTER TABLE tbl\_name ADD FULLTEXT index\_name (column\_list)** − This creates a special FULLTEXT index that is used for text-searching purposes.

The following code block is an example to add index in an existing table.

mysql> ALTER TABLE testalter\_tbl ADD INDEX (c);

You can drop any INDEX by using the **DROP** clause along with the ALTER command.

Try out the following example to drop the above-created index.

mysql> ALTER TABLE testalter\_tbl DROP INDEX (c);

You can drop any INDEX by using the DROP clause along with the ALTER command.

## **ALTER Command to add and drop the PRIMARY KEY**

You can add a primary key as well in the same way. But make sure the Primary Key works on columns, which are NOT NULL.

The following code block is an example to add the primary key in an existing table. This will make a column NOT NULL first and then add it as a primary key.

mysql> ALTER TABLE testalter\_tbl MODIFY i INT NOT NULL;

mysql> ALTER TABLE testalter\_tbl ADD PRIMARY KEY (i);

You can use the ALTER command to drop a primary key as follows −

mysql> ALTER TABLE testalter\_tbl DROP PRIMARY KEY;

To drop an index that is not a PRIMARY KEY, you must specify the index name.

### Displaying INDEX Information

You can use the **SHOW INDEX** command to list out all the indexes associated with a table. The vertical-format output (specified by \G) often is useful with this statement, to avoid a long line wraparound −

Try out the following example −

mysql> SHOW INDEX FROM *table\_name*\G

........

* **Conclusion :**

We have successfully implemented Control Structures & Exception Handling in Fine Calculation on Book Issue.

1. **Sample Code and Output**
2. An employee management system needs to record following data about employees – ID, Name, Age, Department, Salary, Experience, AreaOfExperties. Identify columns, their data types and write create statement. Define primary key.
3. Create a view that will display all details of the employee except Salary and AreaOfExperties.
4. Create a sequence to generate employee id.
5. Create an index for the column ID.
6. Create a synonym for the generated table as “EMP” and demonstrate its use.

INSERT INTO EMPLOYEE VALUES(SEQ.NEXTVAL,'AAA','10','COMP','10000','3','TRAINER');

INSERT INTO EMPLOYEE VALUES(SEQ.NEXTVAL,'BBB','11','CIVIL','20000','4','MANAGER') ;

INSERT INTO EMPLOYEE VALUES(SEQ.NEXTVAL,'CCC','11','IT','30000','5','TEAMLEAD');

----CREATE A VIEW THAT WILL DISPLAY ALL DETAILS OF EMPLOYEE EXCEPT SALARY AND AREAOFEXPERTISE----

CREATE VIEW EMPLOYEE1 AS

SELECT EMPLOYEE\_ID,NAME,AGE,DEPARTMENT,EXPERIENCE FROM EMPLOYEE;

----CREATE A SEQUENCE TO GENERATE EMPLOYEE ID----

CREATE SEQUENCE SEQ

START WITH 1

INCREMENT BY 1;

-----CRAETE AN INDEX FOR COLUMN\_ID-----

CREATE INDEX INDEX\_ID ON EMPLOYEE(EMPLOYEE\_ID);

-----CREATE SYNONYM FOR GENERATED TABLE AS 'EMP' AND DEMONSTRATE ITS USE------

CREATE SYNONYM EMP FOR EMPLOYEE;

SELECT \* FROM EMP;

**Assignment No. A3**

**Problem statement:** Design at least 10 SQL queries for suitable database application using SQL DML statements: Insert, Select, Update, Delete with operators, functions, and set operator.

**Objective:** To implement : Insert, Select, Update, Delete with operators, functions, and set operator

**Solution:**

1. **Related theory**

# AGGREGATION:

MySQL aggregate functions retrieve a single value after performing a calculation on a set of values. In general, aggregate functions ignore null values.

# COUNT:

The COUNT(column name) function returns the number of values (NULL values will not be counted) of the specified column:SQL COUNT(column name)

Syntax

SELECT COUNT(column name) FROM table name;

# ORDER BY CLAUSE:

The ORDER BY keyword is used to sort the result-set by one or more columns. The ORDER BY keyword sorts the records in ascending order by default. To sort the records in a descending order, you can use the DESC keyword.

SQL ORDER BY Syntax:

SELECT \* FROM table name

ORDER BY column name ASC or DESC;

**DISTINCT KEYWORD**:

In a table, a column may contain many duplicate values; and sometimes you only want to list the dfferent (distinct) values. The DISTINCT keyword can be used to return only distinct (different) values.

SQL SELECT DISTINCT Syntax:

SELECT DISTINCT column name,column name FROM table name;

**UNION:**

The UNION operator is used to combine the result-set of two or more SELECT statements. Notice that each SELECT statement within the UNION must have the same number of columns. The columns must also have similar data types. Also, the columns in each SELECT statement must be in the same order.

SQL UNION Syntax

SELECT column name(s) FROM table1

UNION

SELECT column name(s) FROM table2;

**INTERSECTION:**

The SQL INTERSECT clause/operator is used to combine two SELECT statements, but returns rows only from the \_rst SELECT statement that are identical to a row in the second SELECT statement. This means INTERSECT returns only common rows returned by the two SELECT statements.

SQL INTERSECTION Syntax

SELECT column name(s) FROM table1

INTERSECTION

SELECT column name(s) FROM table2;

**IN:**

The IN operator allows you to specify multiple values in a WHERE clause.

SQL IN Syntax

SELECT column name(s)

FROM table name

WHERE column name IN (value1,value2,...);

**NOTIN:**

MySQL NOT IN() makes sure that the expression proceeded does not have any of the values present in the arguments.

SQL NOTIN Syntax:

SELECT column name(s)

FROM table name

WHERE column name NOT IN (value1,value2,...);

Functions performed in banking application are:

1. CREATE BRANCH:

To create a new branch in branch table we use the command:

INSERT into branch values('branch name','branch city',branch assets);

eg: If we want to create a branch named London, having branch city Lawrence and assets of 6000000, we use the command:

INSERT into branch values('London','Lawrence',6000000);

1. CREATE CUSTOMER:

To create a new customer in customer table we use the command:

INSERT into customer values('cust name',cust street','cust city');

eg: If we want to create a branch named London, having branch city Lawrence and assets of 6000000,

we use the command:

INSERT into customer values('John',Main street','Glassgow');

1. SHOW:

To display the details of branch table we use the command:

SELECT \* FROM branch;

To display the details of customer table we use the command

SELECT \* FROM customer;

1. DEPOSIT:

To deposit amout in a particular account we use the command:

UPDATE account set balance = balance + amount WHERE acnt no = account no;

eg: If we want to deposit amount of 25000 in account no 25 we write

the command as:

UPDATE account set balance = balance + 25000

WHERE acnt no = 25;

1. DELETE:

To delete the records of certain customer from customer table we write the command as:

DELETE FROM customer

WHERE cust name='customer name'

eg: If we want to delete the records for the customer named John, we write the command as:

DELETE FROM customer WHERE cust name='John'

1. **Conclusion**

10 SQL queries for suitable database application using SQL DML statements: Insert, Select, Update, Delete with operators, functions, and set operator.

1. **Sample Code and Output**

For the following relation schema:

Account(Acc\_no, branch\_name,balance)

branch(branch\_id, branch\_name,branch\_city,assets)

customer(cust\_id, cust\_name, cust\_street,cust\_city)

Depositor(cust\_id, acc\_no)

Loan(loan\_no, branch\_id, amount)

Borrower(cust\_id, loan\_no)

Create above tables and insert few rows in each table. Solve following query:

1. Find the branches where average account balance > 12000.
2. Find all customers who have an account or loan or both at bank.
3. Find all customers who have both account but not loan at bank.
4. Delete all tuples at every branch located in ‘Nigdi’.
5. Find Maximum loan amount in branch ‘Nigdi’
6. Find no. of depositors at each branch.

For all accounts in Akurdi branch increase the balance by 10%.

CREATE TABLE ACCOUNT

(

ACC\_NO INTEGER,

BRANCH\_NAME VARCHAR(30),

BALANCE INTEGER

);

INSERT INTO ACCOUNT VALUES('10','AKURDI','1000');

INSERT INTO ACCOUNT VALUES('11','RAVET','2000');

INSERT INTO ACCOUNT VALUES('12','CHINCHWAD','3000');

CREATE TABLE BRANCH

(

BRANCH\_ID INTEGER,

BRANCH\_NAME VARCHAR(30),

BRANCH\_CITY VARCHAR(20),

ASSETS VARCHAR(10)

);

INSERT INTO BRANCH VALUES('1','AKURDI','PUNE','HOUSE');

INSERT INTO BRANCH VALUES('2','RAVET','NASHIK','JEWELLERY');

INSERT INTO BRANCH VALUES('3','CHINCHWAD','AMRAVATI','FLAT');

INSERT INTO BRANCH VALUES('4','AKURDI','AMRAVAT','LAT');

INSERT INTO BRANCH VALUES('5','AKURDI','AMRAVA','AT');

INSERT INTO BRANCH VALUES('6','NIGDI','AMRAV','T');

CREATE TABLE CUSTOMER

(

CUST\_ID INTEGER,

CUST\_NAME VARCHAR(30),

CUST\_STREET VARCHAR(20),

CUST\_CITY VARCHAR(10)

);

INSERT INTO CUSTOMER VALUES('20','ABC','LINK ROAD','PUNE');

INSERT INTO CUSTOMER VALUES('21','BCD','LPRO ROAD','NASHIK');

INSERT INTO CUSTOMER VALUES('22','CDE','SHAGUN ROAD','AMRAVATI');

CREATE TABLE DEPOSITOR

(

CUST\_ID INTEGER,

ACC\_NO INTEGER

);

INSERT INTO DEPOSITOR VALUES('20','10');

INSERT INTO DEPOSITOR VALUES('21','11');

INSERT INTO DEPOSITOR VALUES('22','12');

CREATE TABLE LOAN

(

LOAN\_NO INTEGER,

BRANCH\_ID INTEGER,

AMOUNT INTEGER

);

INSERT INTO LOAN VALUES('100','31','10000');

INSERT INTO LOAN VALUES('101','32','20000');

INSERT INTO LOAN VALUES('102','33','30000');

INSERT INTO LOAN VALUES('103','6','90000');

CREATE TABLE BORROWERR

(

CUST\_ID INTEGER,

LOAN\_NO INTEGER

);

INSERT INTO BORROWERR VALUES('41','1');

INSERT INTO BORROWERR VALUES('42','2');

INSERT INTO BORROWERR VALUES('43','3');

--LIST ALL CUSTOMERS IN ALPHABETICAL ORDER WHO HAVE LOAN IN AKURDI BRANCH--

SELECT CUST\_NAME

FROM CUSTOMER,BRANCH

WHERE BRANCH\_NAME='AKURDI'

ORDER BY CUST\_NAME;

---FIND ALL CUSTOMERS WHO HAVE ACCOUNT OR LOAN OR BOTH AT BANK ---

SELECT CUST\_NAME FROM CUSTOMER,DEPOSITOR,BORROWERR

WHERE CUSTOMER.CUST\_ID=DEPOSITOR.CUST\_ID OR BORROWERR.CUST\_ID=CUSTOMER.CUST\_ID;

---FIND ALL CUSTOMERS WHO HAVE BOTH ACCOUNT AND LOAN AT BAMK----

SELECT CUST\_NAME FROM CUSTOMER,DEPOSITOR,BORROWERR

WHERE CUSTOMER.CUST\_ID=DEPOSITOR.CUST\_ID AND BORROWERR.CUST\_ID=CUSTOMER.CUST\_ID;

-----FIND ALL ACCOUNTS IN AKURDI BRANCH INCREASE THE BALANCE BY10%----

UPDATE ACCOUNT

SET BALANCE=BALANCE\*1.1

WHERE BRANCH\_NAME='AKURDI';

-----FIND AVERAGE ACCOUNT BALANCE AT AKURDI BRANCH-------

SELECT AVG(BALANCE) AS "AVERAGE BALANCE" FROM ACCOUNT

WHERE BRANCH\_NAME='AKURDI';

-----FIND MINIMUM ACCOUNT BALANCE AT EACH BRANCH------

SELECT MIN(BALANCE) AS "MINIMUM BALANCE",BRANCH\_NAME FROM ACCOUNT

GROUP BY BRANCH\_NAME;

----DELETE ALL LOAN WITH LOAN AMOUNT BETWEEN 30000 AND 50000-----

DELETE FROM LOAN

WHERE AMOUNT<50000 AND AMOUNT>30000;

---A3(2)---

----FIND ALL BRANCHES WHERE AVERAGE BALANCE IS GREATER THAN 12000---

select BRANCH\_NAME, avg (balance) from account

group by branch\_name

having avg (balance) > 12000;

------FIND ALL CUSTOMERS WHO HAVE ACCOUNT BUT NOT LOAN ----

SELECT CUST\_NAME FROM CUSTOMER,DEPOSITOR,BORROWERR

WHERE CUSTOMER.CUST\_ID=DEPOSITOR.CUST\_ID AND BORROWERR.CUST\_ID!=CUSTOMER.CUST\_ID;

----DELETE ALL TUPLES AT EVERY BRANCH LOCATED IN NIGDI-----

DELETE FROM ACCOUNT

WHERE BRANCH\_NAME='NIGDI';

------FIND MAX LOAN AMOUNT IN NIGDI BRANCH-----

SELECT MAX(LOAN.AMOUNT) AS "MAXIMUM AMOUNT" FROM LOAN,BRANCH

WHERE LOAN.BRANCH\_ID = BRANCH.BRANCH\_ID AND BRANCH.BRANCH\_NAME='NIGDI';

-------FIND NO. OF DEPOSITORS AT EACH BRANCH----

SELECT COUNT(DEPOSITOR.CUST\_ID) AS "NO OF CUSTOMERS",ACCOUNT.BRANCH\_NAME FROM DEPOSITOR,ACCOUNT

WHERE DEPOSITOR.ACC\_NO=ACCOUNT.ACC\_NO

GROUP BY ACCOUNT.BRANCH\_NAME;

**Assignment No. A4**

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

**Objective:** To implement : SQL DML statements: all types of Join, Sub-Query and View

**Solution:**

1. **Related theory**

**SQL Subquery**

**Subquery** or **Inner query** or **Nested query** is a query in a query. SQL subquery is usually added in the WHERE Clause of the SQL statement. Most of the time, a subquery is used when you know how to search for a value using a SELECT statement, but do not know the exact value in the database.

**Subqueries** are an alternate way of returning data from multiple tables.

Subqueries can be used with the following SQL statements along with the comparision operators like =, <, >, >=, <= etc.

* SELECT
* INSERT
* UPDATE
* DELETE

**Different Types of SQL JOINs**

Here are the different types of the JOINs in SQL:

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

**SQL Views**

A VIEW is a virtual table, through which a selective portion of the data from one or more tables can be seen. Views do not contain data of their own. They are used to restrict access to the database or to hide data complexity. A view is stored as a SELECT statement in the database. DML operations on a view like INSERT, UPDATE, DELETE affects the data in the original table upon which the view is based.

**The Syntax to create a sql view is**

CREATE VIEW view\_name   
AS   
SELECT column\_list   
FROM table\_name [WHERE condition];

* ***view\_name*** is the name of the VIEW.
* The SELECT statement is used to define the columns and rows that you want to display in the view.

**For Example:** to create a view on the product table the sql query would be like

CREATE VIEW view\_product   
AS   
SELECT product\_id, product\_name   
FROM product;

1. **Conclusion**

10 SQL queries for suitable database application using SQL DML statements: all types of Join, Sub-Query and View are designed.

1. **Sample Code**
2. **Sample Output**

For the following relation schema:

employee(employee-name, street, city)

works(employee-name, company-name, salary)

company(company-name, city)

manages(employee-name, manager-name)

Create above tables and insert 5 rows in each table. Give an expression in SQL for each of the following queries:

1. Find the names, street address, and cities of residence for all employees who work for 'First Bank Corporation' and earn more than $10,000.
2. Find the names of all employees in the database who live in the same cities as the companies for which they work.
3. Display employee details that live in cities Pune, Mumbai, and Nasik
4. List employees from ‘First Bank Corporation’ that earn salary more than all employees of ‘Small Bank Corporation’
5. Create a view that will display employee details along with name of his/her manager.
6. Find average salary of employees of ‘First Bank Corporation’.

Give employees of ‘First Bank Corporation’ 15% rise if salary is less than 20000.

create table employee(emp\_name VARCHAR(100),street VARCHAR(100) ,city VARCHAR(100));

create table work(name VARCHAR(100),company VARCHAR(100),salary int);

create table company(cname VARCHAR(100),city VARCHAR(100));

create table manages(name VARCHAR(100),manager VARCHAR(100));

insert into employee values('Rohit','Pimpri','Pune');

insert into work values('Rohit','SKF',20000);

INSERT INTO COMPANY VALUES('SKF','Pune');

insert into manages values('Rohit','Tejas');

insert into employee values('Rahul','akurdi','Mumbai');

insert into work values('Rahul','tata',20500);

INSERT INTO COMPANY VALUES('tata','Mumbai');

insert into manages values('Rahul','Rohit');

insert into employee values('Pittu','AKURDI','Pune');

insert into work values('Pittu','RKF',5000);

INSERT INTO COMPANY VALUES('RKF','Pune');

insert into manages values('Pittu','Raj');

--A4(1)

--1

select \* from employee where city='Pune';

--2

SELECT a.name,a.street,a.city FROM EMPLOYEE a,work b WHERE a.name=b.name and b.company='SKF' AND b.salary>30000;

--3

select distinct a.name from employee a,company b where a.city=b.city;

--4

select name from work where company!='SKF';

--5

select name from manages where manager='Tejas';

--6

select avg(salary) from work where company='RKF';

--7

create view mysql as select employee.name,street,city,manager from employee FULL join manages on employee.name = manages.name;

SELECT \* FROM MYsql;

--A4(2)

--1

SELECT a.name,a.street,a.city FROM EMPLOYEE a,work b WHERE a.name=b.name and b.company='SKF' AND b.salary>10000;

--2

select distinct a.name from employee a,company b where a.city=b.city;

--3

select \* from employee where city='Pune' or city='Mumbai' or city='Nashik';

--4

select name from work where COMPANY='tata' and salary > (select max(salary) from work where company='RKF');

--5

create view my as select employee.name,street,city,manager from employee FULL join manages on employee.name = manages.name;

SELECT \* FROM MY;

--6

select avg(salary) from work where company='RKF';

--7

update work SET salary=(1.15\*salary) where company='SKF' and salary<20000

**Assignment No. A5**

**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)

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

**Frame the problem statement for writing PL/SQL block inline with above statement.**

**Objective:**

Design a PL/SQL Ananymous block with exception handling

**Solution:**

1. **Related theory**

PL/SQL is a procedural language that Oracle developed as an extension to standard SQL to provide a way to execute procedural logic on the database.

### PL/SQL Is Block Structured

##### The Syntax for a PL/SQL Block

DECLARE

variable\_declarations

BEGIN

program\_code

EXCEPTION

exception\_handlers

END;

In this syntax, *variable\_declarations* are any variables that you might want to define. Cursor definitions and nested PL/SQL procedures and functions are also defined here. *program\_code* refers to the PL/SQL statements that make up the block. *exception\_handlers* refers to program code that gets triggered in the event of a runtime error or exception.

PL/SQL blocks can be *nested*. One block can contain another block as in the following example:

DECLARE

*variable declarations go here*

BEGIN

*some program code*

BEGIN

*code in a nested block*

*EXCEPTION*

*exception\_handling\_code*

END;

*more program code*

END;

1. **Conclusion**

Designed the PL SQL Block.

1. **Sample Code**

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)

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.

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

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

**Solution**

Declare

mroll number(10);

matt number(10);

Begin

mroll:= &mroll;

select att into matt from stud11 where roll = mroll;

if matt<75 then

dbms\_output.put\_line(mroll||'is detained');

update stud11 set status='D' where roll=mroll;

else

dbms\_output.put\_line(mroll||'is Not detained');

update stud11 set status='ND'where roll=mroll;

end if;

Exception

when no\_data\_found then

dbms\_output.put\_line(mroll||'Not found');

End;

SQL> select \* from stud11;

ROLL ATT STATUS

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

1 70 D

2 75

3 77 ND

4 71

5 79

Enter value for mroll: 6

old 5: mroll:= &mroll;new 5: mroll:= 6;

6Not found

PL/SQL procedure successfully completed.

**Assignment No. A6**

**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.

**Frame the separate problem statement for writing PL/SQL block to implement all types of Cursors inline with above statement. The problem statement should clearly state the requirements.**

**Objective:**

Learn and Implement the cursorse

**Solution:**

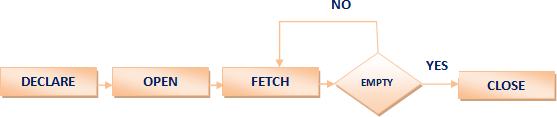
1. **Related theory**

When you work with Oracle database, you work with a complete set of rows returned from an SQL SELECT statement. However the application in some cases cannot work effectively with the entire result set, therefore, the database server needs to provide a mechanism for the application to work with one row or a subset of the result set at a time. As the result, Oracle created PL/SQL cursor to provide these extensions.

A PL/SQL cursor is a pointer that points to the result set of an SQL query against database tables.

## Working with PL/SQL Cursor

The following picture describes steps that you need to follow when you work with a PL/SQL cursor:

[](http://www.plsqltutorial.com/wp-content/uploads/2012/10/plsqlcursor.png)PL/SQL Cursor

Let’s examine each step in greater detail.

### Declaring PL/SQL Cursor

To use PL/SQL cursor, first you must declare it in the declaration section of [PL/SQL block](http://www.plsqltutorial.com/plsql-block-structure/) or in a [package](http://www.plsqltutorial.com/plsql-package/) as follows:

CURSOR cursor\_name [ ( [ parameter\_1 [, parameter\_2 ...] ) ]

      [ RETURN return\_specification ]

IS sql\_select\_statements

[FOR UPDATE [OF [column\_list]];

|  |  |
| --- | --- |
| 1  2  3  4 | CURSOR cursor\_name [ ( [ parameter\_1 [, parameter\_2 ...] ) ]        [ RETURN return\_specification ]  IS sql\_select\_statements  [FOR UPDATE [OF [column\_list]]; |

First, you declare the name of the cursor cursor\_name after the CURSOR keyword. The name of the cursor can have up to 30 characters in length and follows the naming rules of identifiers in PL/SQL. It is important to note that cursor’s name is not a [variable](http://www.plsqltutorial.com/plsql-variables/) so you cannot use it as a variable such as assigning it to other cursor or using it in an expression. The  parameter1*,* parameter2… are optional elements in the cursor declaration. These parameters allow you to pass arguments into the cursor. The RETURN return\_specification is also an optional part.

Second, you specify a valid SQL statement that returns a result set where the cursor points to.

Third, you can indicate a list of columns that you want to update after the FOR UPDATE OF. This part is optional so you can omit it in the CURSOR declaration.

Here is an example of declaring a cursor:

|  |
| --- |
| CURSOR cur\_chief IS        SELECT first\_name,               last\_name,               department\_name        FROM employees e        INNER JOIN departments d ON d.manager\_id = e.employee\_id; |

We retrieved data from employees and departments tables using the SELECTwith the INNER JOIN clause and set the cur\_chief cursor to this result set.

### Opening a PL/SQL Cursor

After declaring a cursor, you can open it by using the

|  |  |
| --- | --- |
| 1 | OPEN cursor\_name [ ( argument\_1 [, argument\_2 ...] ) ]; |

You have to specify the cursor’s name cursor\_name after the keyword OPEN. If the cursor was defined with a parameter list, you need to pass corresponding arguments to the cursor.

When you OPEN the cursor, PL/SQL executes the SQL SELECT statement and identifies the active result set. Notice that the OPEN action does not actually retrieve records from the database. It happens in the FETCH step. If the cursor was declared with the FOR UPDATE clause, PL/SQL locks all the records in the result set.

We can open the cur\_chief cursor as follows:

|  |  |
| --- | --- |
| 1 | OPEN cur\_chief; |

### Fetching Records from PL/SQL Cursor

Once the cursor is open, you can fetch data from the cursor into a [record](http://www.plsqltutorial.com/plsql-record/) that has the same structure as the cursor. Instead of fetching data into a record, you can also fetch data from the cursor to a list of [variables](http://www.plsqltutorial.com/plsql-variables/).

The fetch action retrieves data and fills the record or the variable list. You can manipulate this data in memory. You can fetch the data until there is no record found in active result set.

The syntax of FETCH is as follows:

|  |  |
| --- | --- |
| 1 | FETCH cursor\_name INTO record or variables |

You can test the cursor’s attribute %FOUND or %NOTFOUND to check if the fetch against the cursor is succeeded. The cursor has more  attributes that we will cover in the next section.

We can use [PL/SQL LOOP statement](http://www.plsqltutorial.com/plsql-loop-statement/) together with the FETCH to loop through all records in active result set as follows:

|  |
| --- |
| LOOP      -- fetch information from cursor into record      FETCH cur\_chief INTO r\_chief;        EXIT WHEN cur\_chief%NOTFOUND;        -- print department - chief      DBMS\_OUTPUT.PUT\_LINE(r\_chief.department\_name || ' - ' ||                           r\_chief.first\_name || ',' ||                           r\_chief.last\_name);    END LOOP; |

### Closing PL/SQL Cursor

You should always close the cursor when it is no longer used. Otherwise, you will have a memory leak in your program, which is not expected.

To close a cursor, you use CLOSE statement as follows:

|  |  |
| --- | --- |
| 1 | CLOSE cursor\_name; |

And here is an example of closing the cur\_chief cursor:

|  |  |
| --- | --- |
| 1 | CLOSE cur\_chief; |

## PL/SQL Cursor Attributes

These are the main attributes of a PL/SQL cursor and their descriptions.

| **Attribute** | **Description** |
| --- | --- |
| cursor\_name%FOUND | returns TRUE if record was fetched successfully by cursor cursor\_name |
| cursor\_name%NOTFOUND | returns TRUE if record was not fetched successfully by cursor cursor\_name |
| cursor\_name%ROWCOUNT | returns the number of records fetched from the cursor cursor\_name at the time we test %ROWCOUNT attribute |
| cursor\_name%ISOPEN | returns TRUE if the cursor cursor\_name is open |

1. **Conclusion**

PL/SQL Cursors are studies.

1. **Sample Code**

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

College has decided to mark all those students detained (D) who are having attendance less than 75%. Whenever such update takes place, a record for the same is maintained in the d\_stud table.

create table stud21(roll number(4), att number(4), status varchar(1));

create table d\_stud(roll number(4), att number(4));

**Solution:**

Declare

Cursor crsr\_att is select roll, att,status from stud21 where att<75;

mroll stud21.roll%type;

matt stud21.att%type;

mstatus stud21.status%type;

Begin

open crsr\_att;

if crsr\_att%isopen then

loop

fetch crsr\_att into mroll,matt,mstatus;

exit when crsr\_att%notfound;

if crsr\_att%found then

update stud21 set status='D' where roll=mroll;

insert into d\_stud values(mroll,matt);

end if;

end loop;

end if;

end;

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

Create table new\_class(roll number(10), name varchar(10));

Create table old\_class(roll number(10), name varchar(10));

**Solution:**

Declare

cursor crsr\_class is select \* from old\_class;

cursor crsr\_chk(str\_name varchar) is select roll from new\_class where name = str\_name; str\_roll new\_class.roll%type;

str\_name new\_class.name%type;

v varchar(10);

Begin

Open crsr\_class;

Loop

fetch crsr\_class into str\_roll,str\_name;

Exit When crsr\_class%NOTFOUND;

Open crsr\_chk(str\_name);

Fetch crsr\_chk into v;

if crsr\_chk%FOUND Then

dbms\_output.put\_line('brach'|| str\_name || 'exist');

Else

dbms\_output.put\_line('brach'|| str\_name || ' not exist. Inserting in New\_class table');

insert into new\_class values(str\_roll,str\_name); End if;

Close crsr\_chk;

End loop;

Close crsr\_class;

End;

An explicit cursor FOR LOOP statement prints the last name and job ID of every clerk whose manager has an ID greater than 120.

Create table employees (job\_id number(10), name varchar(10), job varchar(10));

DECLARE

CURSOR c1 IS

SELECT last\_name, job\_id FROM employees

WHERE job\_id LIKE '%CLERK%' AND manager\_id > 120

ORDER BY last\_name;

BEGIN

FOR item IN c1

LOOP

DBMS\_OUTPUT.PUT\_LINE

('Name = ' || item.last\_name || ', Job = ' || item.job\_id);

END LOOP;

END;

**Assignment No. A7**

**Problem statement**: PL/SQL Stored Procedure and Stored Function.

Write a Stored Procedure namely proc\_Grade for the categorization of student. If marks scored by students in examination is <=1500 and marks>=990 then student will be placed in distinction category if marks scored are between 989 and900 category is first class, if marks 899 and 825 category is Higher Second Class

Write a PL/SQL block for using procedure created with above requirement. Stud\_Marks(name, total\_marks) Result(Roll,Name, Class)

**Frame the separate problem statement for writing PL/SQL Stored Procedure and function, inline with above statement. The problem statement should clearly state the requirements.**

**Objective:**

To Create a PL SQL stored procedure and function.

**Solution:**

Theory

The PL/SQL stored procedure or simply a procedure is a PL/SQL block which performs one or more specific tasks. It is just like procedures in other programming languages.

The procedure contains a header and a body.

* **Header:** The header contains the name of the procedure and the parameters or variables passed to the procedure.
* **Body:** The body contains a declaration section, execution section and exception section similar to a general PL/SQL block.

When you want to create a procedure or function, you have to define parameters .There is three ways to pass parameters in procedure:

1. **IN parameters:** The IN parameter can be referenced by the procedure or function. The value of the parameter cannot be overwritten by the procedure or the function.
2. **OUT parameters:** The OUT parameter cannot be referenced by the procedure or function, but the value of the parameter can be overwritten by the procedure or function.
3. **INOUT parameters:** The INOUT parameter can be referenced by the procedure or function and the value of the parameter can be overwritten by the procedure or function.

## PL/SQL Create Procedure

**Syntax for creating procedure:**

1. CREATE [OR REPLACE] PROCEDURE procedure\_name
2. [ (parameter [,parameter]) ]
3. IS
4. [declaration\_section]
5. BEGIN
6. executable\_section
7. [EXCEPTION
8. exception\_section]
9. END [procedure\_name];

# PL/SQL Function

The PL/SQL Function is very similar to PL/SQL Procedure. The main difference between procedure and a function is, a function must always return a value, and on the other hand a procedure may or may not return a value. Except this, all the other things of PL/SQL procedure are true for PL/SQL function too.

**Syntax to create a function:**

1. CREATE [OR REPLACE] FUNCTION function\_name [parameters]
2. [(parameter\_name [IN | OUT | IN OUT] type [, ...])]
3. RETURN return\_datatype
4. {IS | AS}
5. BEGIN
6. < function\_body >
7. END [function\_name];

**Here:**

* **Function\_name:** specifies the name of the function.
* **[OR REPLACE]** option allows modifying an existing function.
* The **optional parameter list** contains name, mode and types of the parameters.
* **IN** represents that value will be passed from outside and OUT represents that this parameter will be used to return a value outside of the procedure.

**The function must contain a return statement.**

* RETURN clause specifies that data type you are going to return from the function.
* Function\_body contains the executable part.
* The AS keyword is used instead of the IS keyword for creating a standalone function.

1. **Conclusion**

Created PL/SQL function and procedure.

1. **Sample Code and Output**

Write a Stored Procedure namely proc\_Grade for the categorization of student. If marks scored by students in examination is <=1500 and marks>=990 then student will be placed in distinction category if marks scored are between 989 and900 category is first class, if marks 899 and 825 category is Higher Second Class

Write a PL/SQL block for using procedure created with above requirement.

Stud\_Marks(name, total\_marks)

Result(Roll,Name, Class)

**Solution**

create or replace procedure proc\_grade

(temp in number,

p\_roll\_no out stud\_marks.roll\_no%type,

p\_name out stud\_marks.name%type,

p\_total out stud\_marks.total\_marks%type)

as

begin

select name,total\_marks,roll\_no into p\_name,p\_total,p\_roll\_no from stud\_marks where roll\_no=temp;

if p\_total <=1500 and p\_total >= 990 then

insert into result values(p\_roll\_no,p\_name,'distinction');

else if p\_total <=989 and p\_total >= 900 then

insert into result values(p\_roll\_no,p\_name,'first class');

else if p\_total <=899 and p\_total >= 825 then

insert into result values(p\_roll\_no,p\_name,'HSC');

else

insert into result values(p\_roll\_no,p\_name,'fail');

end if;

end if;

end if;

exception

when no\_data\_found then

dbms\_output.put\_line('Roll no ' || temp ||' not found');

end;

/

**Execution**

procedure created.

**SQL> select \* from result;**

**no rows selected**

**Calling Procedure**

Declare

temp number(20);

p\_roll\_no stud\_marks.roll\_no%type;

p\_name stud\_marks.name%type;

p\_total stud\_marks.total\_marks%type;

Begin

temp:=&temp;

Proc\_grade(temp,p\_roll\_no,p\_name,p\_total);

End;

/

**Enter value for temp: 1**

**old 7: temp:=&temp;**

**new 7: temp:=1;**

**PL/SQL procedure successfully completed.**

**SQL> select \* from result;**

ROLL\_NO NAME CLASS

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

1 ABC distinction

**SQL> select \* from stud\_marks;**

ROLL\_NO NAME TOTAL\_MARKS

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

1 ABC 1000

2 XYZ 960

3 PQR 850

4 LMN 820

Write a function namely func\_Grade for the categorization of student. If marks scored by students in examination is <=1500 and marks>=990 then student will be placed in distinction category if marks scored are between 989 and900 category is first class, if marks 899 and 825 category is Higher Second Class

Write a PL/SQL block for using function created with above requirement.

Stud\_Marks(name, total\_marks)

Result(Roll,Name, Class)

**Solution**

create or replace function fun\_grade

(temp in number)

return number

as

p\_roll\_no stud\_marks.roll\_no%type;

p\_name stud\_marks.name%type;

p\_total stud\_marks.total\_marks%type;

begin

select name,total\_marks,roll\_no into p\_name,p\_total,p\_roll\_no from stud\_marks where roll\_no=temp;

if p\_total <=1500 and p\_total >= 990 then

insert into result values(p\_roll\_no,p\_name,'distinction');

else if p\_total <=989 and p\_total >= 900 then

insert into result values(p\_roll\_no,p\_name,'first class');

else if p\_total <=899 and p\_total >= 825 then

insert into result values(p\_roll\_no,p\_name,'HSC');

else

insert into result values(p\_roll\_no,p\_name,'fail');

end if;

end if;

end if;

return p\_roll\_no;

exception

when no\_data\_found then

dbms\_output.put\_line('Roll no ' || temp ||' not found');

end;

Function created.

**SQL> select \* from stud\_marks;**

ROLL\_NO NAME TOTAL\_MARKS

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

1 ABC 1000

2 XYZ 960

3 PQR 850

4 LMN 820

**SQL> select \* from result;**

ROLL\_NO NAME CLASS

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

1 ABC distinction

2 XYZ first class

3 PQR HSC

**Calling Function:**

Declare

temp number(20):=&temp;

p\_roll\_no varchar2(20);

Begin

p\_roll\_no :=fun\_grade(temp);

End;

/

Enter value for temp: 4

old 2: temp number(20):=&temp;

new 2: temp number(20):=4;

**PL/SQL procedure successfully completed.**

SQL> select \* from result;

ROLL\_NO NAME CLASS

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

1 ABC distinction

2 XYZ first class

3 PQR HSC

4 LMN fail

**Assignment No. A8**

**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.

**Frame the problem statement for writing Database Triggers of all types, in-line with above statement. The problem statement should clearly state the requirements.**

**Objective:**

To learn and implement the PL SQL Triggers

**Solution:**

1. **Related theory**

# PL/SQL Trigger

Trigger is invoked by Oracle engine automatically whenever a specified event occurs.Trigger is stored into database and invoked repeatedly, when specific condition match.

Triggers are stored programs, which are automatically executed or fired when some event occurs.

Triggers are written to be executed in response to any of the following events.

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

## Advantages of Triggers

These are the following advantages of Triggers:

* Trigger generates some derived column values automatically
* Enforces referential integrity
* Event logging and storing information on table access
* Auditing
* Synchronous replication of tables
* Imposing security authorizations
* Preventing invalid transactions

## Creating a trigger:

**Syntax for creating trigger:**

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;

**Here,**

* CREATE [OR REPLACE] TRIGGER trigger\_name: It creates or replaces an existing trigger with the trigger\_name.
* {BEFORE | AFTER | INSTEAD OF} : This specifies when the trigger would 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 would 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, like INSERT, UPDATE, and DELETE.
* [FOR EACH ROW]: This specifies a row level trigger, i.e., the trigger would 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.

1. **Conclusion**

Databasetriggers has been studied

**Assignment No:B1**

* **Title:** Study of Open Source NOSQL Database: MongoDB (Installation, Basic CRUD operations, Execution)
* **Objectives**: To perform installation and CRUD operations of MongoDB.
* **Problem Statement** :To Study and implement database with suitable example using MongoDB and implement all basic operations.
* **Hardware and Software Requirement**:

1. Computer System with Linux/ Open Source Operating System
2. MongoDB

* **Theory :**

*Introduction*: MongoDB is an open-source,cross platform,document oriented,database that provides high performance,high availability and automatic scaling.

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

*Collections*: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.

*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.

*Advantages of MongoDB* :

* Schema less − MongoDB is a document database in which one collection holds different documents. Number of fields, content and size of the document can differ from one document to another.
* Structure of a single object is clear.
* No complex joins.
* Deep query-ability. MongoDB supports dynamic queries on documents using a document-based query language that's nearly as powerful as SQL.
* Ease of scale-out − MongoDB is easy to scale.
* Conversion/mapping of application objects to database objects not needed.
* Uses internal memory for storing the (windowed) working set, enabling faster access of data.

*Installation*: To start MongoDB using two-tier:

TERMINAL1:

student@student-OptiPlex-390$:cdmongodb-linux-x8664-2.6.3/bin

student@student-OptiPlex-390:/mongodb-linux-x8664-2.6.3/bin$./mongod--dbpath’/home/student/mongo’

Now start with the second terminal bypressing ctrl+shift+t

TERMINAL2:

./mongo 'ipaddressofserver’

*Basic CRUD Operations:*

CREATE:

It is used to create a database with specified database name and if it already exists, it will return the existing one.

Syntax:

USE databasename

DROP:

It is used to drop the database as specified if it exists. If database name not is specified, current database is dropped.

Syntax:

>db.dropDatabse()

CREATE COLLECTION:

After stating the database to be used, a collection is created inside a database. In mongodb you don’t need to create collection.MongoDB creates collection automatically, when you insert some document.

Syntax:

db.createCollection(name,options)

INSERT DOCUMENT:

For insertion of documents in collection,insert()or save()methods are used.

Syntax:

>db.collectionname.insert(document)

UPDATE DOCUMENT

It is used to update the existing documents in the collection. It won’t adda new key value.

Syntax:

>db.collectionname.update(selectioncriteria,updatedata)

PRETTY METHOD

To display the result in a formatted way,pretty() method can be used. It is used along with the find() methods.

Syntax:

>db.collectionname.find().pretty()

* **Coding** :

1.Create a database:

use mydb

switched to db mydb

2.Drop a database:

use mydb

switched to db mydb

db.dropDatabase()

{”dropped”:”mydb”,”ok”:1}

3.Create a Collection:

db.createCollection(”mycol”,capped: true,autoIndexID:true,size: 6142800,max:10000)

”ok”:1

4.Insert a document:

db.mycol.insert({id:ObjectId(7df78ad8902c),title:’MongoDBOverview’,description:’MongoDBisnosqldatabase’,by:’tutorialspoint’,url:’[http://www.tutorialspoint.com](http://www.tutorialspoint.com/)’,tags:[’mongodb’,’database’,’NoSQL’],likes:100})

5.Update Document:

db.mycol.update({’title’:’MongoDBOverview’},{$set:{’title’:’NewMongoDB Tutorial’}})

6.Prettymethod

db.mycol.find().pretty()

{“id”:ObjectId(7df78ad8902c),

“title”: “MongoDBOverview”

“description”:”MongoDB is no sql database”,

“by”: “tutorialspoint”,

“url”: "[hHYPERLINK "http://www.tutorialspoint.com/"tHYPERLINK "http://www.tutorialspoint.com/"tp://www.tutorialspoint.com](http://www.tutorialspoint.com/)”,

“tags”:[“mongodb”,“database”,“NoSQL”], “likes”:”100”

}

* **Test Cases:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test\_id** | **Description** | **Expected O/P** | **Actual O/P** | **Result** |
| 1 | To create a database mydb | Creation of mydb database | switched to db mydb. | Pass |
| 2 | To drop a database | Database “mydb”should be removed. | {”dropped”:”mydb”,”ok”:1} | Pass |
| 3 | To Create a Collection | A Collection named mycol should be created | ”ok”:1 | pass |
| 4. | To insert a document | New document should be inserted. | New document is inserted | pass |
| 5. | To update a document | “MongoDB Overview” should be updated. | Updated Name :’NewMongoDB Tutorial’}}) | pass |

* **Conclusion**:

We have studied Open Source NOSQL Database: MongoDB (Installation, Basic CRUD operations, Execution)

**Sample Code**

Create a collection **student** in mongodb with fields (stud\_id, stud\_name, dept, marks)

1. Insert few documents in the collection
2. Display details of all students in the collection
3. Display details of students in ‘Computer’ department
4. Update marks of student with id =10. Change the marks to 70
5. Delete student with id =15

1.insert into collection:-

db.student.insert({"stud\_id":"1","name":"Shubh","dept":"comp","marks":"1000"})

db.student.insert({"stud\_id":"2","name":"saroj","dept":"comp","marks":"999"})

db.student.insert({"stud\_id":"41","name":"Shubhash","dept":"comp","marks":"1000"})

insert many:-

db.student.insert([{"stud\_id":"4","name":"Tej","dept":"comp","marks":"1100"},{"stud\_id":"44","name":"hero","dept":"mech","marks":"1000"},{"stud\_id":"55","name":"rider","dept":"comp","marks":"1000"}])

2.Display Details:-

db.student.find()

3.Display dept=comp students:-

db.student.find({"dept":"comp"})

4.update entry:-

db.student.update({"stud\_id":"1"},{$set:{"marks":"70"}})

5.delete entry:-

db.student.remove({"stud\_id":"2"})

Create a collection **employee** in mongodb with fields (emp\_id, emp\_name, dept, salary)

1. Insert few documents in the collection
2. Display details of all employees in the collection
3. Display details of employees in ‘HR’ department
4. Update salary of employees with id =1001. Change the salary to 30000
5. Delete employees with id =1005

1.insert:-

db.emp.insert([{"emp\_id":"1","name":"Shubh","dept":"comp","salary":"10005"},{"emp\_id":"2","name":"Shubhash","dept":"hr","salary":"10000"},

{"emp\_id":"3","name":"baji","dept":"stack","salary":"10120"},

{"emp\_id":"4","name":"ram","dept":"comp","salary":"10120"}])

2.all employees:-

db.emp.find()

3.dept=hr:-

db.emp.find({"dept":"hr"})

4.update salary of emp with id...

db.emp.update({"emp\_id":"2"},{$set:{"salary":"30000"}})

5.delete entry:

**Assignment No. :B2**

* **Title**:Design and Develop MongoDB Queries using CRUD operations. (Use CRUD operations, SAVE method, logical operators)
* **Objectives**: To implement SAVE and logical operators of MongoDB.
* **Problem** **Statement** :To implement SAVE method and logical AND AND logical OR operators using MongoDB.
* **Hardware and Software Requirement**:

1. Computer System with Linux/ Open Source Operating System
2. MongoDB

* **Theory**:

MongoDB's update() and save() methods are used to update document into a collection. The update() method updates the values in the existing document while the save() method replaces the existing document with the document passed in save() method.

*MongoDB Update() Method :*

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

Syntax

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

>db.COLLECTION\_NAME.update(SELECTION\_CRITERIA, UPDATED\_DATA)

*MongoDB Save() Method* :

The save() method replaces the existing document with the new document passed in the save() method.

Syntax

The basic syntax of MongoDB save() method is shown below −

>db.COLLECTION\_NAME.save({\_id:ObjectId(),NEW\_DATA})

**Logical Operators in MongoDB**:

*AND in MongoDB*

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

Following is the basic syntax of AND −

>db.mycol.find(

{

$and: [

{key1: value1}, {key2:value2}

]

}

).pretty()

*OR in MongoDB*

To query documents based on the OR condition, you need to use $or keyword

Following is the basic syntax of OR −

>db.mycol.find(

{

$or: [

{key1: value1}, {key2:value2}

]

}

).pretty()

* **Coding:**

1.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"}

>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})

2.Example :

>db.mycol.save( {

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

"by":"Tutorials Point" } )

>db.mycol.find()

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

"by":"Tutorials Point"}

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

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

**Logical Operators**:

*Logical AND*

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

"\_id": ObjectId(7df78ad8902c),

"title": "MongoDB Overview",

"description": "MongoDB is no sql database",

"by": "tutorials point",

"url": "<http://www.tutorialspoint.com>",

"tags": ["mongodb", "database", "NoSQL"],

"likes": "100"

}

For the above given example, equivalent where clause will be ' where by = 'tutorials point' AND title = 'MongoDB Overview' '.

You can pass any number of key, value pairs in find clause.

*Logical OR*

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

{

"\_id": ObjectId(7df78ad8902c),

"title": "MongoDB Overview",

"description": "MongoDB is no sql database",

"by": "tutorials point",

"url": "<http://www.tutorialspoint.com>",

"tags": ["mongodb", "database", "NoSQL"],

"likes": "100"

}

*Using AND and OR Together*

Example

The following example will show the documents that have likes greater than 10 and whose title is either 'MongoDB Overview' or by is

'tutorials point'. Equivalent SQL where clause is 'where likes>10 AND (by = 'tutorials point' OR title = 'MongoDB Overview')'

>db.mycol.find({"likes": {$gt:10}, $or: [{"by": "tutorials point"},

{"title": "MongoDB Overview"}]}).pretty()

{

"\_id": ObjectId(7df78ad8902c),

"title": "MongoDB Overview",

"description": "MongoDB is no sql database",

"by": "tutorials point",

"url": "<http://www.tutorialspoint.com>",

"tags": ["mongodb", "database", "NoSQL"],

"likes": "100"

* **Test Cases :**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test\_id** | **Description** | **Expected O/P** | **Actual O/P** | **Result** |
| 1 | To SAVE a new document | Existing document should be replaced by new one | Existing document is replaced by new one | Pass |
| 2 | AND | db.mycol.find({$and:[{"by":"tutorials point"},{"title": "MongoDB Overview"}]}). | Any one of the two field is inserted | Pass |
| 3 | OR | db.mycol.find({$or:[{"by":"tutorials point"},{"title": "MongoDB Overview"}]}). | Both the fields are inserted | pass |

* **Conclusion**:

We have implemented UPDATE method, SAVE method and logical operators using MongoDB.

**Sample Code**

Create a collection **student** in mongodb and insert few documents with fields (stud\_id, stud\_name, dept, marks)

1. Find students having marks greater than 50
2. Find students having marks between 50 and 80
3. Find students having marks more than 60 from ‘Computer’ department
4. Update marks of all students from ‘Civil’ department. Set marks to 30
5. Delete students from ‘Chemical’ department having marks less than 30

db.stud.insert({"stud\_id":"2","name":"saroj","dept":"comp","marks":999})

db.stud.insert([{"stud\_id":"4","name":"Tej","dept":"comp","marks":100},{"stud\_id":"44","name":"hero","dept":"mech","marks":1000},{"stud\_id":"55","name":"rider","dept":"comp","marks":1010}])

1. marks greater than 900:-

db.stud.find({"marks":{$gt:900}})

2.marks between 900 to 1000

db.stud.find({"marks":{$gte:900,$lte:1000}})

3.marks greater than 900 and dept=comp:-

db.stud.find({$and:[{"marks":{$gt:900}},{"dept":"comp"}]})

4. update marks=100 where branch=mech

db.employee.update({"dept":"comp"},{$set:{"salary":"40000"}})

5.delete stud from mech with marks less than 950

db.employee.remove({$and:[{"dept":"hr"},{"salary":{$lt:45000}}]})

Create a collection **employee** in mongodb and insert few documents with fields (emp\_id, emp\_name, dept, salary)

1. Find employees having salary greater than 50000
2. Find employees having salary between 50000 and 80000
3. Find employees having salary more than 60000 from ‘HR’ department
4. Update marks of all students from ‘Civil’ department. Set marks to 30

Delete students from ‘Chemical’ department having marks less than 30

db.emp1.insert([{"emp\_id":"1","name":"Shubh","dept":"comp","salary":10005},{"emp\_id":"2","name":"Shubhash","dept":"hr","salary":10000},

{"emp\_id":"3","name":"baji","dept":"stack","salary":10120},

{"emp\_id":"4","name":"ram","dept":"comp","salary":10120}])

1.salary grater then 5000

db.employee.find({"salary":{$gt:50000}})

2.10000 to 10050

db.employee.find({"salary":{$gte:50000,$lte:80000}})

3.salary>6000 & dept=hr

db.employee.find({$and:[{"salary":{$gt:60000}},{"dept":"hr"}]})

**Assignmen No.B3**

* **Title**:Implement aggregation and indexing with suitable example using MongoDB.

* **Objectives**: To implement Aggregation and Indexing using MongoDB.
* **Problem** **Statement** : To perform MongoDB Database programming to aggregate and index a document.
* **Hardware and Software Requirement**:

1. Computer System with Linux/ Open Source Operating System
2. MongoDB

* **Theory**:

Indexing:

Indexes support the efficient resolution of queries. Without indexes, MongoDB must scan every document of a collection to select those documents that match the query statement. This scan is highly inefficient and require MongoDB to process a large volume of data.

Indexes are special data structures, that store a small portion of the 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 as specified in the index.

The ensureIndex() Method

To create an index you need to use ensureIndex() method of MongoDB.

Syntax

The basic syntax of ensureIndex() method is as follows().

>db.COLLECTION\_NAME.ensureIndex({KEY:1})

Here key is the name of the field on which you want to create index and 1 is for ascending order. To create index in descending order you need to use -1.

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.

The aggregate() Method

For the aggregation in MongoDB, you should use aggregate() method.

Basic syntax of aggregate() method is as follows −

>db.COLLECTION\_NAME.aggregate(AGGREGATE\_OPERATION)

**Coding:**

*Indexing*

Example

>db.mycol.ensureIndex({"title":1})

>

In ensureIndex() method you can pass multiple fields, to create index on multiple fields.

>db.mycol.ensureIndex({"title":1,"description":-1})

*Aggregation*:

Example

In the collection you have the following data −

{

\_id: ObjectId(7df78ad8902c)

title: 'MongoDB Overview',

tags: description: 'MongoDB is no sql database',

by\_user: 'tutorials point',

url: '<http://www.tutorialspoint.com> ['mongodb', 'database', 'NoSQL'],

likes: 100

},

{

\_id: ObjectId(7df78ad8902d)

title: 'NoSQL Overview',

description: 'No sql database is very fast',

by\_user: 'tutorials point',

url: '<http://www.tutorialspoint.com>',

tags: ['mongodb', 'database', 'NoSQL'],

likes: 10

},

{

\_id: ObjectId(7df78ad8902e)

title: 'Neo4j Overview',

description: 'Neo4j is no sql database',

by\_user: 'Neo4j',

url: '<http://www.neo4j.com>',

tags: ['neo4j', 'database', 'NoSQL'],

likes: 750

},

Now from the above collection, if you want to display a list stating how many tutorials are written by each user, then you will use the following aggregate() method −

> db.mycol.aggregate([{$group : {\_id : "$by\_user", num\_tutorial : {$sum : 1}}}])

{

"result" : [

{

"\_id" : "tutorials point",

"num\_tutorial" : 2

},

{

"\_id" : "Neo4j",

"num\_tutorial" : 1

}

],

Output:

"ok" : 1

1.$sum :Sums up the defined value from all documents in the collection.

db.mycol.aggregate([{$group : {\_id : "$by\_user",num\_tutorial : {$sum : "$likes"}}}])

2.$avg : Calculates the average of all given values from all documents in the collection.

db.mycol.aggregate([{$group : {\_id : "$by\_user", num\_tutorial : {$avg : "$likes"}}}])

3. $min : Gets the minimum of the corresponding values from all documents in the collection. db.mycol.aggregate([{$group : {\_id : "$by\_user", num\_tutorial : {$min : "$likes"}}}])

4. $max :Gets the maximum of the corresponding values from all documents in the collection. db.mycol.aggregate([{$group : {\_id : "$by\_user", num\_tutorial : {$max : "$likes"}}}])

* **Test Cases :**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test\_id** | **Description** | **Expected O/P** | **Actual O/P** | **Result** |
| 1 | Indexing | If ( +1) then indexing is done in ascending order.  If ( -1) then indexing is done in descending order. | Indexing is done | Pass |
| 2 | Aggregation | db.mycol.aggregate([{$group : {\_id : "$by\_user", num\_tutorial : {$sum : 1}}}]) | {  "\_id" : "tutorials point",  "num\_tutorial" : 2  },  {  "\_id" : "Neo4j",  "num\_tutorial" : 1  }  ], | Pass |

* **Conclusion**:

We have successfully implemented indexing and aggregation method using MongoDB.

**Sample Code**

Create a collection **employee** in mongodb and insert few documents with fields (emp\_id, emp\_name, dept, salary)

1. Display maximum salary in each department
2. Display minimum salary in each department
3. Display average salary in each department
4. Display number of employees in each department

> db.createCollection("employee");

{ "ok" : 1 }

> db.employees.insert(

... [

... {

... emp\_id:1,

... emp\_name:"ABC",

... dept:"HR",

... salary:20000

... },

... {

... emp\_id:2,

... emp\_name:"BCD",

... dept:"Developer",

... salary:25000

... },

... {

... emp\_id:3,

... emp\_name:"CDE",

... dept:"Testing",

... salary:10000

... },

... {

... emp\_id:4,

... emp\_name:"DEF",

... dept:"Developer",

... salary:20000

... },

{

... emp\_id:5,

... emp\_name:"EFG",

... dept:"Testing",

... salary:40000

... },

{

... emp\_id:6,

... emp\_name:"FGH",

... dept:"HR",

... salary:33000

... }

... ]);

BulkWriteResult({

    "writeErrors" : [ ],

    "writeConcernErrors" : [ ],

    "nInserted" : 4,

    "nUpserted" : 0,

    "nMatched" : 0,

    "nModified" : 0,

    "nRemoved" : 0,

    "upserted" : [ ]

})

> db.employee.find().pretty();

{

    "\_id" : ObjectId("59ef675a0c0ab723f41f6f1c"),

    "emp\_id" : 1,

    "name" : "ABC",

    "dept" : "HR",

    "salary" : 20000

}

{

    "\_id" : ObjectId("59ef675a0c0ab723f41f6f1d"),

    "emp\_id" : 2,

    "name" : "BCD",

    "dept" : "Developer",

    "salary" : 25000

}

{

    "\_id" : ObjectId("59ef675a0c0ab723f41f6f1e"),

    "emp\_id" : 3,

    "name" : "Cde",

    "dept" : "Testing",

    "salary" : 10000

}

{

    "\_id" : ObjectId("59ef675a0c0ab723f41f6f1f"),

    "emp\_id" : 4,

    "name" : "Bdq",

    "dept" : "Developer",

    "salary" : 20000

}

1. find max salary in each dept

> db.employee.aggregate(

... [

... {

... $group:{

... \_id:"$dept",

... max\_salary:{$max:"$salary"}

... }

... }

... ]);

{ "\_id" : "Developer", "max\_salary" : 25000 }

{ "\_id" : "Testing", "max\_salary" : 10000 }

{ "\_id" : "HR", "max\_salary" : 20000 }

2.find min salary in each department

> db.employee.aggregate(

... [

... {

... $group:{

... \_id:"$dept",

... min\_salary:{$min:"$salary"}

... }

... }

... ]);

{ "\_id" : "Developer", "min\_salary" : 20000 }

{ "\_id" : "Testing", "min\_salary" : 10000 }

{ "\_id" : "HR", "min\_salary" : 20000 }

3.display average salary in each department

> db.employee.aggregate(

... [

... {

... $group:{

... \_id:"$dept",

... avg\_salary:{$avg:"$salary"}

... }

... }

... ]);

{ "\_id" : "Developer", "avg\_salary" : 22500 }

{ "\_id" : "Testing", "avg\_salary" : 10000 }

{ "\_id" : "HR", "avg\_salary" : 20000 }

4.Count of all the employees in departments

> db.employee.aggregate(

... [

... {

... $group:{

... \_id:"$dept",

... no\_of\_emp:{$sum:1}

... }

... }

... ]);

{ "\_id" : "Developer", "no\_of\_emp" : 2 }

{ "\_id" : "Testing", "no\_of\_emp" : 1 }

{ "\_id" : "HR", "no\_of\_emp" : 1 }

Create a collection **bank** in mongodb and insert few documents with fields (cust\_id, cust\_name, branch, balance)

1. Display maximum balance in each branch
2. Display minimum balance in each branch
3. Display average balance in each branch
4. Display number of customers in each branch

db.bank.insert([{cust\_id:1, cust\_name:"Arun",branch:"Akurdi", balance:10000},{cust\_id:2, cust\_name:"Sharvari",branch:"chinchwad", balance:50000},{cust\_id:3, cust\_name:"Ganesh",branch:"chinchwad", balance:15000},{cust\_id:4, cust\_name:"Aditya",branch:"pune", balance:5000},{cust\_id:5,

2.min balance in each branch:-

db.bank.aggregate([{$group:{\_id:"$branch",min\_balance:{$min:"$balance"}}}])

3.average balance in each branch:-

db.bank.aggregate([{$group:{\_id:"$branch",avg\_balance:{$avg:"$balance"}}}])

4.number of customers in each branch:-

db.bank.aggregate([{$group:{\_id:"$branch",no\_of\_cust:{$sum:1}}}])

$group:{

... \_id:"$dept",

... no\_of\_emp:{$sum:1}

**Assignmen No.B4**

* **Title**:Implement Map reduces operation with suitable example using MongoDB.
* **Objectives**: To implement map reduce using MongoDB.
* **Problem** **Statement** :To perform mapReduce operation on records to reduce database size using MongoDB.
* **Hardware and Software Requirement**:

1. Computer System with Linux/ Open Source Operating System
2. MongoDB

* **Theory**:

As per the MongoDB documentation, Map-reduce is a data processing paradigm for condensing large volumes of data into useful aggregated results. MongoDB uses mapReduce command for map-reduce operations. MapReduce is generally used for processing large data sets.

MapReduce Command

Following is the syntax of the basic mapReduce command −

>db.collection.mapReduce( function() {emit(key,value);}, //map function

function(key,values) {return reduceFunction}, { //reduce function

out: collection,

query: document,

sort: document,

limit: number

}

)

The map-reduce function first queries the collection, then maps the result documents to emit key-value pairs, which is then reduced based on the keys that have multiple values.

In the above syntax –

* map is a javascript function that maps a value with a key and emits a key-value pair
* reduce is a javascript function that reduces or groups all the documents having the same key
* out specifies the location of the map-reduce query result
* query specifies the optional selection criteria for selecting documents
* sort specifies the optional sort criteria
* limit specifies the optional maximum number of documents to be returned.
* **Coding:**:

Using MapReduce

Consider the following document structure storing user posts. The document stores user\_name of the user and the status of post.

{

"post\_text": "tutorialspoint is an awesome website for tutorials",

"user\_name": "mark",

"status":"active"

}

Now, we will use a mapReduce function on our posts collection to select all the active posts, group them on the basis of user\_name and then count the number of posts by each user using the following code −

>db.posts.mapReduce(

function() { emit(this.user\_id,1); },

function(key, values) {return Array.sum(values)}, {

query:{status:"active"},

out:"post\_total"

}

)

The above mapReduce query outputs the following result −

{

"result" : "post\_total",

"timeMillis" : 9,

"counts" : {

"input" : 4,

"emit" : 4,

"reduce" : 2,

"output" : 2},

"ok" : 1,

}

**Output**:

To see the result of this mapReduce query, use the find operator −

>db.posts.mapReduce(

function() { emit(this.user\_id,1); },

function(key, values) {return Array.sum(values)}, {

query:{status:"active"},

out:"post\_total"

}

).find()

The above query gives the following result which indicates that both users tom and mark have two posts in active states −

{ "\_id" : "tom", "value" : 2 }

{ "\_id" : "mark", "value" : 2 }

* **Test** **Cases** :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test\_id** | **Description** | **Expected O/P** | **Actual O/P** | **Result** |
| 1 | MapReduce Operation | MapReduce function on our posts collection to select all the active posts, group them on the basis of user\_name and then count the number of posts by each user | { "\_id" : "tom", "value" : 2 }  { "\_id" : "mark", "value" : 2 } | Pass |

* **Conclusion**:

We have successfully implemented Mapreduce method using MongoDB.

**Sample Code**

Create a collection **bank** in mongodb and insert few documents with fields (cust\_id, cust\_name, branch, balance)

Write a MapReduce function to display balance in each branch of the bank

cust\_id, cust\_name, branch, balance

db.bank.insert([{"cust\_id":1,"cust\_name":"Tejas","branch":"Akurdi","bal":30000},

{"cust\_id":2,"cust\_name":"Tejal","branch":"Akurdi","bal":35000},

{"cust\_id":3,"cust\_name":"Amey","branch":"Pimpri","bal":3000},

{"cust\_id":4,"cust\_name":"Ajay","branch":"Pimpri","bal":10000},

{"cust\_id":5,"cust\_name":"Ashish","branch":"Pune","bal":50000}])

Mapreduce:-

var Mapfunction = function(){emit(this.branch,this.bal)}

var Reducefunction = function(key,values){return Array.sum(values)}

db.bank.mapReduce(Mapfunction,Reducefunction,{'out':'resultant'})

Answer:-

db.resultant.find()

Create a collection books in mongodb and insert few documents with fields (book\_id, title, author, type)

Write a MapReduce function to display number of books of each type

book\_id, title, author, type

db.books.insert([{book\_id:1,title:"My",author:"Rajesh",type:"songs"}])

db.books.insert([{book\_id:2,title:"Jack",author:"Raj",type:"Poem"},

{book\_id:3,title:"What",author:"John",type:"Story"},

{book\_id:4,title:"Real",author:"Warner",type:"Real Stories"}])

db.books.insert([{book\_id:5,title:"Ram",author:"Raj",type:"Poem"},

{book\_id:6,title:"Temperature",author:"Tejas",type:"Story"}])

Mapreduce:-

var Mapfunction = function(){emit(this.type,1)}

var Reducefunction = function(key,values){return Array.sum(values)}

db.books.mapReduce(Mapfunction,Reducefunction,{'out':'typeofbooks'})

Answer:-

db.typeofbooks.find()

**Assignmen No.B5**

* **Title**:Design and Implement any 5 queries using MongoDB.
* **Objectives**:To implement MongoDB queries for various purposes.
* **Problem** **Statement** :To implement remove,greater than,less than,sort, limit and skip methods using MongoDB.
* **Hardware and Software Requirement**:

1. Computer System with Linux/ Open Source Operating System
2. MongoDB

* **Theory**:

*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 just One 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)

*Conditional* *Queries*:

To query the document on the basis of some condition, you can use following operations.

Equality

{<key>: <value>} db.mycol.find({"by":"tutorials point"}).pretty()

Less than

{$lt: <value>}} db.mycol.find({"likes": {$lt:50}}).pretty()

Less Than Equals

{<key>: {$lte: <value>}} db.mycol.find({"likes": {$lte:50}}).pretty()

Greater Than {<key>: {$gt: <value>}} db.mycol.find({"likes": {$gt:50}}).pretty()

Greater Than Equals {<key>: {$gte: <value>}} db.mycol.find({"likes": {$gte:50}}).pretty()

Not Equals

{<key>: {$ne: <value>}} db.mycol.find({"likes": {$ne:50}}).pretty() where likes != 50

*Limit*

The Limit() Method

To limit the records in MongoDB, you need to use limit() method. The method accepts one number type argument, which is the number of documents that you want to be displayed.

Syntax

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

>db.COLLECTION\_NAME.find().limit(NUMBER)

*Skip*

The Skip() method

Apart from limit() method, there is one more method skip() which also accepts number type argument and is used to skip the number of documents.

Syntax

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

>db.COLLECTION\_NAME.find().limit(NUMBER).skip(NUMBER)

*Sort*

The sort() Method

To sort documents in MongoDB, you need to use sort() method. The method accepts a document containing a list of fields along with their sorting order. To specify sorting order 1 and -1 are used. 1 is used for ascending order while -1 is used for descending order.

Syntax

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

>db.COLLECTION\_NAME.find().sort({KEY:1})

* **Coding:**

1.Remove:

Consider the mycol collection has the following data.

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

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

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

Following example will remove all the documents whose title is 'MongoDB Overview'.

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

>db.mycol.find()

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

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

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()

2.Limit :

Consider the collection myycol 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 display only two documents while querying the document.

>db.mycol.find({},{"title":1,\_id:0}).limit(2)

{"title":"MongoDB Overview"}

{"title":"NoSQL Overview"}

>

If you don't specify the number argument in limit() method then it will display all documents from the collection.

3.Sort :

Example

Consider the collection myycol 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 display the documents sorted by title in the descending order.

>db.mycol.find({},{"title":1,\_id:0}).sort({"title":-1})

{"title":"Tutorials Point Overview"}

{"title":"NoSQL Overview"}

{"title":"MongoDB Overview"}

>

Please note, if you don't specify the sorting preference, then sort() method will display the documents in ascending order.

* **Test Cases** :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test\_id** | **Description** | **Expected O/P** | **Actual O/P** | **Result** |
| 1 | Remove | >db.mycol.remove({'title':'MongoDB Overview'})  MongoDB overview should be removed. | { "\_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}  { "\_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point Overview"} | Pass |
| 2 | Limit | db.mycol.find({},{"title":1,\_id:0}).limit(2)  will display only two documents while querying the document. | {"title":"MongoDB Overview"}  {"title":"NoSQL Overview"} | Pass |
| 3 | Sort | display the documents sorted by title in the descending order.  >db.mycol.find({},{"title":1,\_id:0}).sort({"title":-1}) | {"title":"Tutorials Point Overview"}  {"title":"NoSQL Overview"}  {"title":"MongoDB Overview"}   * With ids 7,6,5 resp | pass |

* **Conclusion**:

We have successfully implemented remove,limit,skip,sort and conditional methods using MongoDB.

**Sample Code**

Create collection Student in mongodb and insert few documents having fields (student\_id, student\_name, city, marks, subjects). The filed subject will accept array of subjects.

1. Display student information with ID=10
2. Display students information with city = ‘Pune’ and Marks = 50
3. Delete student with ID=15
4. Display students with marks between 50 and 90
5. Display student information having maximum marks
6. Update value of city = ‘Nagpur’ for student with ID = 15

student\_id, student\_name, city, marks, subjects

db.stude.insert({"stud\_id":1,"name":"Tejas","city":"mumbai","marks":90,"sub":"all"})

db.stude.insert([{"stud\_id":2,"name":"Tej","city":"pune","marks":95,"sub":"bio"},

{"stud\_id":3,"name":"Teja","city":"pimpri","marks":80,"sub":"geography"}])

db.stude.insert({"stud\_id":4,"name":"rohan","city":"pune","marks":50,"sub":"bio"})

1. Display student information with ID=3

db.stude.find({"stud\_id":3})

2. Display students information with city = ‘Pune’ and Marks = 50

db.stude.find({$and:[{"city":"pune"},{"marks":50}]})

3. Delete student with ID=15

db.stude.remove({"stud\_id":2})

4. Display students with marks between 50 and 90

db.stude.find({"marks":{$gte:50,$lte:90}})

5. Display student information having maximum marks

db.stude.find().sort({"marks":-1}).limit(1)

6. Update value of city = ‘Nagpur’ for student with ID = 4

db.stude.update({"stud\_id":4},{$set:{"city":"Nagpur"}})

Create collection employee in mongodb and insert few documents with fields (id, name, department, salary)

1. Display list of employees from ‘HR’ department
2. Display list of employees from ‘Admin’ having salary greater than 20000
3. Update salary of all employees in ‘Admin’ department to 25000
4. Display min and max salary from ‘Development’ department
5. Display average salary of each department
6. Delete student with ID=15

1. employees from department = hr

db.emp1.find({"dept":"hr"})

2. Display list of employees from ‘hr’ having salary greater than 6000

db.emp1.find({$and:[{"salary":{$gt:6000}},{"dept":"hr"}]})

3. Update salary of all employees in ‘Admin’ department to 25000

db.emp1.update({"dept":"Admin"},{$set:{"salary":25000}},{multi:true})

4. Display min and max salary from ‘comp’ department

db.emp1.aggregate([{$group:{\_id:"stack",max:{$max:"$salary"},min:{$min:"$salary"}}}])

5.

db.emp1.aggregate([{$group:{\_id:"$dept",avg:{$avg:"$salary"}}}])

6.

db.emp1.remove({id:15})

**Assignmen No.C1**

* **Title**: Write a program to implement MogoDB database connectivity with PHP/ python/Java Implement Database navigation operations (add, delete, edit etc. ) using ODBC/JDBC.
* **Objectives**:To establish a connectivity between MongoDB and Java.
* **Problem** **Statement** :Implement MongoDB database connectivity with Java .

Implement Database navigation operation (add , delete ,edit) using JDBC.

* **Theory**:

Java JDBC is a java API to connect and execute query with the database. JDBC API uses jdbc drivers to connect with the database.

**Solution:**

1. **Related theory**

Before we start using MongoDB in our Java programs, we need to make sure that we have MongoDB JDBC Driver and Java set up on the machine. You can check Java tutorial for Java installation on your machine. Now, let us check how to set up MongoDB JDBC driver.

* You need to download the jar from the path Download mongo.jar. Make sure to download latest release of it.
* You need to include the mongo.jar into your classpath.

**Connect to database**

To connect database, you need to specify database name, if database doesn't exist then mongodb creates it automatically.

Code snippets to connect to database would be as follows:

import com.mongodb.MongoClient;

import com.mongodb.MongoException;

import com.mongodb.WriteConcern;

import com.mongodb.DB;

import com.mongodb.DBCollection;

import com.mongodb.BasicDBObject;

import com.mongodb.DBObject;

import com.mongodb.DBCursor;

import com.mongodb.ServerAddress;

import java.util.Arrays;

public class MongoDBJDBC{

public static void main( String args[] ){

try{

// To connect to mongodb server

MongoClient mongoClient = new MongoClient( "localhost" , 27017 );

// Now connect to your databases

DB db = mongoClient.getDB( "test" );

System.out.println("Connect to database successfully");

boolean auth = db.authenticate(myUserName, myPassword);

System.out.println("Authentication: "+auth);

}catch(Exception e){

System.err.println( e.getClass().getName() + ": " + e.getMessage() );

}

}

}

Now, let's compile and run above program to create our database test. You can change your path as per your requirement. We are assuming current version of JDBC driver mongo-2.10.1.jar is available in the current path

$javac MongoDBJDBC.java

$java -classpath ".:mongo-2.10.1.jar" MongoDBJDBC

Connect to database successfully

Authentication: true

If you are going to use Windows machine, then you can compile and run your code as follows:

$javac MongoDBJDBC.java

$java -classpath ".;mongo-2.10.1.jar" MongoDBJDBC

Connect to database successfully

Authentication: true

Value of **auth** will be true, if the user name and password are valid for the selected database.

**Create a collection**

To create a collection, **createCollection()** method of **com.mongodb.DB** class is used.

Code snippets to create a collection:

Now connect to your databases

DB db = mongoClient.getDB( "test" );

System.out.println("Connect to database successfully");

DBCollection coll = db.createCollection("mycol");

System.out.println("Collection created successfully");

**Getting/ selecting a collection**

To get/select a collection from the database, **getCollection()** method of **com.mongodb.DBCollection** class is used.

DBCollection coll = db.getCollection("mycol");

System.out.println("Collection mycol selected successfully");

**Insert a document**

To insert a document into mongodb, **insert()** method of **com.mongodb.DBCollection** class is used.

BasicDBObject doc = new BasicDBObject("title", "MongoDB").

append("description", "database").

append("likes", 100).

append("url", "http://www.tutorialspoint.com/mongodb/").

append("by", "tutorials point");

coll.insert(doc);

System.out.println("Document inserted successfully");

**Retrieve all documents**

To select all documents from the collection, **find()** method of **com.mongodb.DBCollection** class is used. This method returns a cursor, so you need to iterate this cursor.

DBCursor cursor = coll.find();

int i=1;

while (cursor.hasNext()) {

System.out.println("Inserted Document: "+i);

System.out.println(cursor.next());

i++;

}

**Update document**

To update document from the collection, **update()** method of **com.mongodb.DBCollection** class is used.

// find hosting = hostB, and update the clients to 110

BasicDBObject newDocument = new BasicDBObject();

newDocument.put("clients", 110);

BasicDBObject searchQuery = new

BasicDBObject().append("hosting", "hostB");

collection.update(searchQuery, newDocument,false,true);

**To find perticular document**

To find document from the collection, you need to select the documents using **Find or findOne()** method and then pass document to be serached as argument to it.

DBObject myDoc = coll.findOne(searchQuery);

**Delete first document**

To delete first document from the collection, you need to first select the documents using **findOne()** method and then **remove** method of **com.mongodb.DBCollection** class.

DBObject myDoc = coll.findOne();

col1.remove(myDoc);

1. **Conclusion**

By using Mongo.Jar file we can connect java and mongoDb and can perform all the operation using java connectivity

**Sample Code**

import com.mongodb.\*;

public class Mongo {

public static void main( String args[] ) {

try{

MongoClient mongoClient = new MongoClient( "localhost" , 27017 );

DB db = mongoClient.getDB( "test" );

System.out.println("Connect to database successfully");

DBCollection col1=db.createCollection("shree1",new BasicDBObject());

System.out.println("collection created");

DBCollection col2=db.createCollection("shruti2",new BasicDBObject());

System.out.println("collection created");

DBCollection col1=db.getCollection("shree1");

System.out.println("collection created");

BasicDBObject doc1 = new BasicDBObject();

doc1.put("name", "shraddha");

doc1.put("website", "google.com");

BasicDBObject doc2 = new BasicDBObject();

doc2.put("addressLine1", "Sweet Home");

doc2.put("addressLine2", "Karol Bagh");

doc2.put("addressLine3", "New Delhi, India");

col1.insert(new BasicDBObject[] {doc1,doc2});

}catch(Exception e){

System.err.println( e.getClass().getName() + ": " + e.getMessage() );

}

}

}

/\*output

> show dbs

PL01 0.078GB

abc 0.078GB

cat 0.078GB

doc 0.078GB

local 0.078GB

maggi 0.078GB

mdb 0.078GB

suj 0.078GB

test 0.078GB

xyz 0.078GB

> use doc

switched to db doc

> show collections

shree1

shruti2

> db.shree1.find()

{ "\_id" : ObjectId("57ac2983522ed0bb4f416f15"), "name" : "shraddha", "website" : "google.com" }

{ "\_id" : ObjectId("57ac2983522ed0bb4f416f16"), "addressLine1" : "Sweet Home", "addressLine2" : "Karol Bagh", "addressLine3" : "New Delhi, India" }

\*/

**Assignmen No.C2**

* **Title**: Implement MYSQL/Oracle database connectivity with PHP/ python/Java Implement Database navigation operations (add, delete, edit,) using ODBC/JDBC.
* **Objectives**:To establish a connectivity between MYSQL and Java.To manipulate MYSQL commands using Java.
* **Problem** **Statement** :Implement MYSQL database connectivity with Java .

Implement Database navigation operation (add , delete ,edit) using JDBC.

* **Theory**:

Java JDBC is a java API to connect and execute query with the database. JDBC API uses jdbc drivers to connect with the database.

**JDBC Driver :**

**J**DBC Driver is a software component that enables java application to interact with the database. There are 4 types of JDBC drivers:

* JDBC-ODBC bridge driver
* Native-API driver (partially java driver)
* Network Protocol driver (fully java driver)
* Thin driver (fully java driver)

Steps to connect to the database in java:

**1) Register the driver class**

|  |
| --- |
| The forName() method of Class class is used to register the driver class. This method is used to dynamically load the driver class. |

***Syntax of forName() method***

* public static void forName(String className)throws ClassNotFoundException

***Example to register the MYSQLDriver class***

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

***2) Create the connection object***

|  |
| --- |
| The getConnection() method of DriverManager class is used to establish connection with the database. |

***Syntax of getConnection() method***

* public static Connection getConnection(String url)throws SQLException
* static Connection getConnection(String url,String name,String password)
* throws SQLException

***Example to establish connection with the Oracle database***

* Connection con=DriverManager.getConnection(
* "jdbc:mysql://localhost:3306/sonoo","root","password");

**3) *Create the Statement object***

|  |
| --- |
| The createStatement() method of Connection interface is used to create statement. The object of statement is responsible to execute queries with the database. |

***Syntax of createStatement() method***

* public Statement createStatement()throws SQLException

***Example to create the statement object***

* Statement stmt=con.createStatement();

**4) Execute the query**

|  |
| --- |
| The executeQuery() method of Statement interface is used to execute queries to the database. This method returns the object of ResultSet that can be used to get all the records of a table. |

***Syntax of executeQuery() method***

* public ResultSet executeQuery(String sql)throws SQLException

***Example* *to* *execute* *query***

* ResultSet rs=stmt.executeQuery("select \* from emp");
* while(rs.next()){
* System.out.println(rs.getInt(1)+" "+rs.getString(2));  }

**5) *Close the connection object***

|  |
| --- |
| By closing connection object statement and ResultSet will be closed automatically. The close() method of Connection interface is used to close the connection. |

***Syntax of close() method***

* public void close()throws SQLException

***Example to close connection***

* con.close();

Statement interface

The **Statement interface** provides methods to execute queries with the database. The statement interface is a factory of ResultSet i.e. it provides factory method to get the object of ResultSet.

***Commonly used methods of Statement interface:***

The important methods of Statement interface are as follows:

|  |
| --- |
| **1) public ResultSet executeQuery(String sql):**is used to execute SELECT query. It returns the object of ResultSet. |
| **2) public int executeUpdate(String sql):**is used to execute specified query, it may be create, drop, insert, update, delete etc. |
| **3) public boolean execute(String sql):**is used to execute queries that may return multiple results. |
| **4) public int[] executeBatch():**is used to execute batch of commands. |

ResultSet interface

The object of ResultSet maintains a cursor pointing to a row of a table. Initially, cursor points to before the first row.

But we can make this object to move forward and backward direction by passing either TYPE\_SCROLL\_INSENSITIVE or TYPE\_SCROLL\_SENSITIVE in createStatement(int,int) method as well as we can make this object as updatable by:

Statement stmt = con.createStatement(ResultSet.TYPE\_SCROLL\_INSENSITIVE,

                     ResultSet.CONCUR\_UPDATABLE);

**Commonly used methods of ResultSet interface**

|  |  |
| --- | --- |
| **1) public boolean next():** | is used to move the cursor to the one row next from the current position. |
| **2) public boolean previous():** | is used to move the cursor to the one row previous from the current position. |
| **3) public boolean first():** | is used to move the cursor to the first row in result set object. |
| **4) public boolean last():** | is used to move the cursor to the last row in result set object. |
| **5) public boolean absolute(int row):** | is used to move the cursor to the specified row number in the ResultSet object. |
| **6) public boolean relative(int row):** | is used to move the cursor to the relative row number in the ResultSet object, it may be positive or negative. |
| **7) public int getInt(int columnIndex):** | is used to return the data of specified column index of the current row as int. |
| **8) public int getInt(String columnName):** | is used to return the data of specified column name of the current row as int. |
| **9) public String getString(int columnIndex):** | is used to return the data of specified column index of the current row as String. |
| **10) public String getString(String columnName):** | is used to return the data of specified column name of the current row as String. |

PreparedStatement interface

The PreparedStatement interface is a subinterface of Statement. It is used to execute parameterized query.

Let's see the example of parameterized query:

String sql="insert into emp values(?,?,?)";

As you can see, we are passing parameter (?) for the values. Its value will be set by calling the setter methods of PreparedStatement.

**Why use PreparedStatement?**

**Improves performance**: The performance of the application will be faster if you use PreparedStatement interface because query is compiled only once.

***How to get the instance of PreparedStatement?***

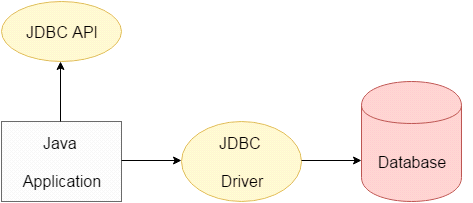
The prepareStatement() method of Connection interface is used to return the object of PreparedStatement. Syntax:

public PreparedStatement prepareStatement(String query)throws SQLException{}

**Methods of PreparedStatement interface**

The important methods of PreparedStatement interface are given below:

|  |  |
| --- | --- |
| **Method** | **Description** |
| public void setInt(int paramIndex, int value) | sets the integer value to the given parameter index. |
| public void setString(int paramIndex, String value) | sets the String value to the given parameter index. |
| public void setFloat(int paramIndex, float value) | sets the float value to the given parameter index. |
| public void setDouble(int paramIndex, double value) | sets the double value to the given parameter index. |
| public int executeUpdate() | executes the query. It is used for create, drop, insert, update, delete etc. |
| public ResultSet executeQuery() | executes the select query. It returns an instance of ResultSet. |



* **Coding:**

**Example to Connect Java Application with mysql database**

import java.sql.\*;

class MysqlCon{

public static void main(String args[]){

try{

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

Connection con=DriverManager.getConnection(

"jdbc:mysql://localhost:3306/databasename","root","root");

Statement stmt=con.createStatement();

ResultSet rs=stmt.executeQuery("select \* from emp");

while(rs.next())

System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getString(3));

con.close();

}catch(Exception e){ System.out.println(e);}

}

}

**Source Code :**

import java.util.\*;

import java.sql.\*;

class Studsys{

public static void main(String args[]){

Scanner input=new Scanner(System.in);

int choice;

try{

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

Connection con=DriverManager.getConnection(

"jdbc:mysql://localhost:3306/system?useSSL=false","root","1234");

boolean count;

Statement stmt=con.createStatement();

stmt.execute("use system");

stmt.execute("drop table studinfo");

stmt.execute("create table studinfo(Rollno int(3) auto\_increment,primary key(Rollno),Name varchar(15),Clg\_name varchar(15))");

do

{

System.out.println("\t\t\*\*\*\*\*\*\*\*\*\*STUDENTS SYSTEM\*\*\*\*\*\*\*\*\*\*\*\*");

System.out.println("\t\t1)CREATE RECORDS");

System.out.println("\t\t2)INSERT RECORD");

System.out.println("\t\t3)DELETE RECORD");

System.out.println("\t\t4)DISPLAY RECORDS");

System.out.println("\t\t5)MODIFY RECORD");

System.out.println("\t\t6)SEARCH RECORD");

System.out.println("\t\t7)EXIT");

System.out.println("\tEnter your choice: ");

choice=input.nextInt();

int roll\_no;

String name,clg;

switch(choice)

{

case 1:

int n;

System.out.println("\nEnter the no of records u want to create: ");

n=input.nextInt();

try{

for(int i=0;i<n;i++)

{

System.out.println("\nEnter roll no: ");

roll\_no=input.nextInt();

System.out.println("Enter name: ");

name=input.next();

System.out.println("Enter clg name: ");

clg=input.next();

stmt.execute("insert into studinfo values("+roll\_no+",'"+name+"','"+clg+"')");

System.out.println("\nData inserted!!!!");

}

}

catch(SQLException ex){

System.out.println("Exception caught!! it is: "+ex.getMessage()); }

break;

case 2:

System.out.println("\nEnter roll no: ");

roll\_no=input.nextInt();

System.out.println("Enter name: ");

name=input.next();

System.out.println("Enter clg name: ");

clg=input.next();

try{

stmt.execute("insert into studinfo values("+roll\_no+",'"+name+"','"+clg+"')");

System.out.println("\nData inserted!!!!"); }

catch(SQLException ex){

System.out.println("\nException Caught!!!It is: "+ex.getMessage());

}

break;

case 3:

System.out.println("\n\t1)ROLL NO\n\t2)NAME\n\t3)COLLEGE\n\t\tDELETE BY: ");

int dch=input.nextInt();

switch(dch)

{

case 1:

System.out.println("\nEnter the roll no of record to be deleted: ");

int droll=input.nextInt();

try{

count= stmt.execute("delete from studinfo where Rollno="+droll+"");

if(count==true)

System.out.println("\nRecord Deleted!!!!");

else

System.out.println("\nRecord cannot be Deleted as Record not present!!!!");

}

catch(SQLException ex){

System.out.println("Exception caught!! it is: "+ex.getMessage()); }

break;

case 2:

System.out.println("\nEnter the name of record to be deleted: ");

String dname=input.next();

try{

count=stmt.execute("delete from studinfo where Name='"+dname+"'");

if(count==true)

System.out.println("\nRecord Deleted!!!!");

else

System.out.println("\nRecord cannot be Deleted as Record not present!!!!");

}

catch(SQLException ex){

System.out.println("Exception caught!! it is: "+ex.getMessage()); }

break;

case 3:

System.out.println("\nEnter the College name of record to be deleted: ");

String dclg=input.next();

try{

count=stmt.execute("delete from studinfo where Clg\_name='"+dclg+"'");

if(count==true)

System.out.println("\nRecord Deleted!!!!");

else

System.out.println("\nRecord cannot be Deleted as Record not present!!!!");

}

catch(SQLException ex){

System.out.println("Exception caugh!! it is: "+ex.getMessage()); }

break;

}

break;

case 4:

ResultSet rs=stmt.executeQuery("select \* from studinfo order by Rollno");

while(rs.next())

System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getString(3));

break;

case 5:

System.out.println("\n\t1)ROLL NO\n\t2)NAME\n\t3)COLLEGE\n\t\tMODIFY BY: ");

int mch=input.nextInt();

switch(mch)

{

case 1:

System.out.println("\nEnter the roll no of record to be modified: ");

int mroll=input.nextInt();

System.out.println("\nEnter the new roll no : ");

int nroll=input.nextInt();

try{

count= stmt.execute("update studinfo set Rollno="+nroll+" where Rollno="+mroll+"");

if(count==true)

System.out.println("\nRecord Modified!!!!");

else

System.out.println("\nRecord cannot be Modified as Record is not present!!!!");

}

catch(SQLException ex){

System.out.println("Exception caught!! it is: "+ex.getMessage()); }

break;

case 2:

System.out.println("\nEnter the name of record to be modified: ");

int mname=input.nextInt();

System.out.println("\nEnter the new roll no : ");

int nname=input.nextInt();

try{

count=stmt.execute("update studinfo set Name='"+nname+"' where Name='"+mname+"'");

if(count==true)

System.out.println("\nRecord modified!!!!");

else

System.out.println("\nRecord cannot be modified as Record not present!!!!");

}

catch(SQLException ex){

System.out.println("Exception caught!! it is: "+ex.getMessage()); }

break;

case 3:

System.out.println("\nEnter the college name of record to be modified: ");

int mclg=input.nextInt();

System.out.println("\nEnter the new roll no : ");

int nclg=input.nextInt();

try{

count=stmt.execute("update studinfo set Clg\_name='"+nclg+"' where Clg\_name='"+mclg+"'");

if(count==true)

System.out.println("\nRecord modified!!!!");

else

System.out.println("\nRecord cannot be modified as Record not present!!!!");

}

catch(SQLException ex){

System.out.println("Exception caught!! it is: "+ex.getMessage()); }

break;

}

break;

case 6:

System.out.println("\n\t1)ROLL NO\n\t2)NAME\n\t3)COLLEGE\n\t\nSEARCH BY: ");

int sch=input.nextInt();

switch(sch)

{

case 1:

System.out.println("\nEnter the roll no of record to be searched: ");

int sroll=input.nextInt();

try{

count= stmt.execute("select \* from studinfo where Rollno="+sroll+"");

if(count==true){

System.out.println("\nRecord found!!!!");

ResultSet rss=stmt.executeQuery("select \* from studinfo where Rollno='"+sroll+"'");

while(rss.next())

System.out.println(rss.getInt(1)+" "+rss.getString(2)+" "+rss.getString(3));

}

else

System.out.println("\nRecord not present!!!!");

}

catch(SQLException ex){

System.out.println("Exception caught!! it is: "+ex.getMessage()); }

break;

case 2:

System.out.println("\nEnter the name of record to be searched: ");

int sname=input.nextInt();

try{

count= stmt.execute("select \* from studinfo where Name='"+sname+"'");

if(count==true){

System.out.println("\nRecord found!!!!");

ResultSet r=stmt.executeQuery("select \* from studinfo where Name='"+sname+"'");

while(r.next())

System.out.println(r.getInt(1)+" "+r.getString(2)+" "+r.getString(3));

}

else

System.out.println("\nRecord not present!!!!");

}

catch(SQLException ex){

System.out.println("Exception caught!! it is: "+ex.getMessage()); }

break;

case 3:

System.out.println("\nEnter the College name of record to be searched: ");

int sclg=input.nextInt();

try{

count= stmt.execute("select \* from studinfo where Clg\_name='"+sclg+"'");

if(count==true){

System.out.println("\nRecord found!!!!");

ResultSet rr=stmt.executeQuery("select \* from studinfo where Clg\_name='"+sclg+"'");

while(rr.next())

System.out.println(rr.getInt(1)+" "+rr.getString(2)+" "+rr.getString(3));

}

else

System.out.println("\nRecord not present!!!!");

}

catch(SQLException ex){

System.out.println("Exception caught!! it is: "+ex.getMessage()); }

break;

}

break;

}

}while(choice != 7);

con.close();

}catch(Exception e){ System.out.println(e);}

}

}

* **Conclusion :**

We have successfully implemented MYSQL Database connectivity with JAVA .