GOVERNMENT COLLEGE OF ENGINEERING, JALGAON

(An Autonomous Institute of Government of Maharashtra)
National Highway No.6, JALGAON – 425 002; **Phone** – 0257-2281522 **Fax** – 0257-2281319
E-Mail – princoej@rediffmain.com **Web.** – http://gcoej.ac.in

---

Year and Programme: Third Year B. Tech.(Computer) ***Academic Year: 2019-20***

Course Code and Course Name: CO356 Database Management System Lab

INDEX

Sr. No.	Title of Experiment	Date of Performance	Date of Completion	Sign of Teacher	Page No.
1	Conversion of ER to Relational Schema and DDL queries	/ /2020	/ /2020		
2	DML Queries	/ /2020	/ /2020		
3	TCL and DCL Queries	/ /2020	/ /2020		
4	Views, triggers and Indexing	/ /2020	/ /2020		
5	DML queries using PL/SQL	/ /2020	/ /2020		
6	Installation Of MongoDB	/ /2020	/ /2020		
7	Create collection and insert document using Python	/ /2020	/ /2020		
8	Perform Queries on collection.	/ /2020	/ /2020		
9	Perform update and delete operations on a collection	/ /2020	/ /2020		
10	Perform Aggregation function on collection.	/ /2020	/ /2020		
11	MINI PROJECT	/ /2020	/ /2020		

CERTIFICATE

This is to certify that Mr. /Miss._____PRN_____of T. Y. B. Tech. ***(Computer Engineering.)*** has satisfactorily completed the experiments/work/ assignments specified for Internal Continuous Assessment of ***C*O356 Database Management System Lab** as specified in syllabus of this Institute for the academic year 2019-20.

Date: / /2020

Course Teacher **Course Coordinator** **HoD** **Principal**

PRACTICAL NO. 01

AIM : Map the ER/EER diagrams to a relational schema. Be sure to underline all primary keys, include all necessary foreign keys and indicate referential integrity constraints. Create database of the same schema using Data Definition Language (DDL). Use all DDL statements (Create, Alter, Drop) with all possible options and constraints (Primary key, Foreign Key, unique, Not Null, Default, Check etc.)

THEORY

Data: Data is distinct piece of information, usually formatted in a special way. Data can exists in a variety of forms-as number or text. Example: name of person such as abc, or any number 123,etc.

Database: Database is a collection of data. It contains information about one particular enterprise. Example of enterprise and it's database are-Bank:-which stores customers banking data.

DBMS: Database Management System (DBMS) is a collection of interrelated data and a set of programs to access the data. Examples are-Oracle, Microsoft Access, etc.

Schema: The overall design of the database is called database schemas. Example: A database schema corresponds to the programming language type defination.

Relation: A database relation is a predefined row/column format for storing information in a relational database. Relations are equivalent to tables.

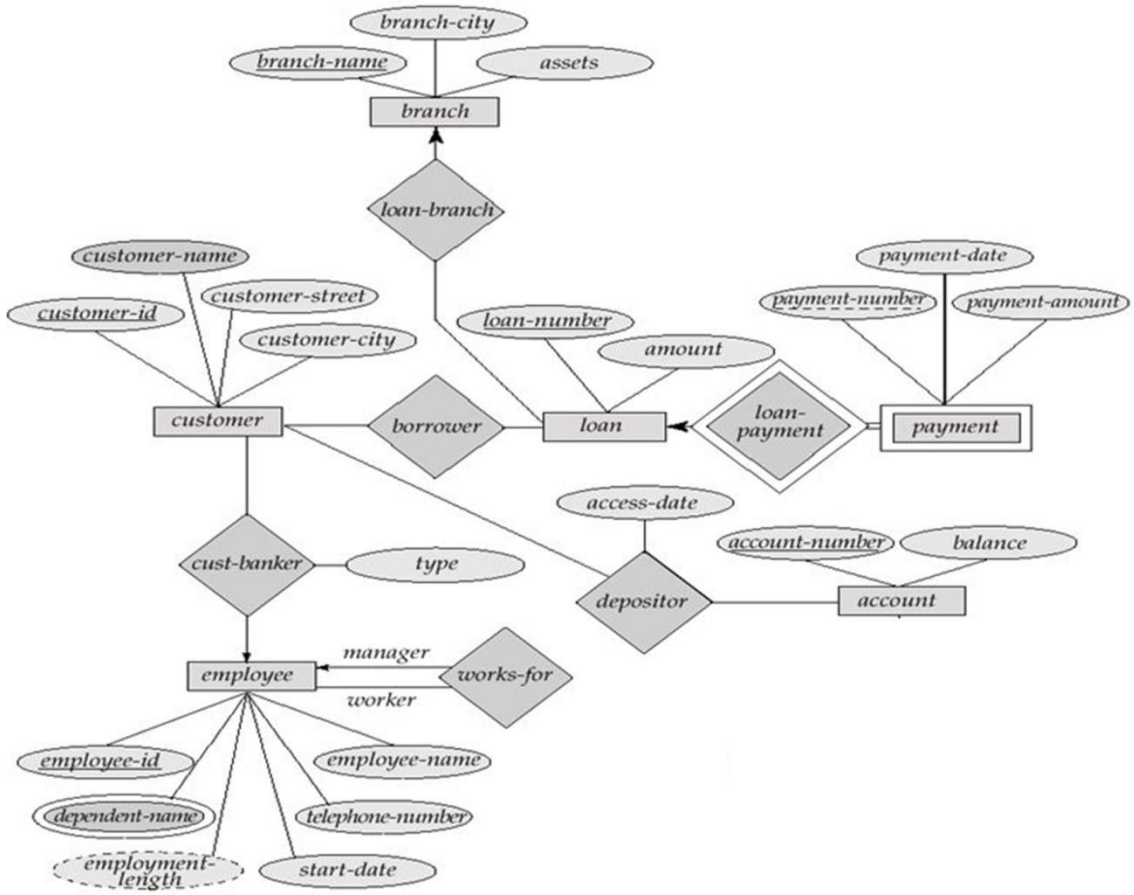
Record/ Tuple: In the context of a relational database, a row—also called a record or tuple—represents a single, implicitly structured data item in a table.

Attribute/Field/Data item: In general, an attribute is a characteristic. In a database management system (DBMS), an attribute refers to a database component, such as table. It also may refer to a database field. Attributes describe the instances in the row of a database.

Different datatypes used in SQL

VARCHAR2(size [BYTE | CHAR]) Variable length character string having maximum length size bytes or characters. Maximum size is 4000 bytes or characters, and minimum is 1 byte or 1 character. You must specify size for VARCHAR2. BYTE indicates that the column will have byte length semantics. CHAR indicates that the column will have character semantics.

ER-diagram



Relational Schema

Customer(customer_id,customer_name,customer_street,customer_city)
Loan(loan_number,amount)
Borrower(customer_id,loan_number)
branch(branch_name,branch_city,assets)
loan_branch(loan_number,branch_name)
employee(employee-id,employee-name,employee-member,start-date,dependent-name,employee-length)
depositor(customer_id,access_date,account_number)
account(account_number,balance)
loan_payment(loan_number,payment_number)
payment(payment_number,payment_date,payment_amount)

DDL: (Create Alter, Drop)

The Create Table Command: - it defines each column of the table uniquely. Each column has minimum of three attributes, a name , data type and size.

Syntax: Create table <table name> (<col1> <datatype>(<size>),<col2> <datatype><size>));

Ex: create table emp(empno number(4) primary key, ename char(10));

2. Modifying the structure of tables.

a)add new columns

Syntax: Alter table <tablename> add(<new col><datatype(size),<new col>datatype(size));

Ex:alter table emp add(sal number(7,2));

3. Dropping a column from a table.

Syntax: Alter table <tablename> drop column <col>;

Ex: alter table emp drop column sal;

4. Modifying existing columns.

Syntax: Alter table <tablename> modify(<col><newdatatype>(<newsize>));

Ex: alter table emp modify(ename varchar2(15));

5. Renaming the tables

Syntax: Rename <oldtable> to <new table>;

Ex:rename emp to emp1;

6. truncating the tables.

Syntax: Truncate table <tablename>;

Ex: trunc table emp1;

7. Destroying tables.

Syntax: Drop table <tablename>;

Ex: drop table emp;

Constraints (primary key, foreign key, unique, Not Null, check,default)

DDL COMMAND

It is used to communicate with database. DDL is used to Create an object, to Alter the structure of an object, to drop the object created.

The commands used are: Create, Alter, Drop, Truncate

INTEGRITY CONSTRAINT

An integrity constraint is a mechanism used by oracle to prevent invalid data entry into the table. It has enforcing the rules for the columns in a table. The types of the integrity

constraints are:

a) Domain Integrity

b) Entity Integrity

c) Referential Integrity

1. **Domain Integrity Constraint**

This constraint sets a range and any violations that take place will prevent the user from performing the manipulation at caused the breach. It includes:

Not Null constraint:

While creating tables, by default the rows can have null value .the enforcement of not null constraint in a table ensure that the table contains values.

Principle of null values:

- Setting null value is appropriate when the actual value is unknown, or when a value would not be meaningful.
- A null value is not equivalent to a value of zero.
- A null value will always evaluate to null in any expression.
- When a column name is defined as not null, that column becomes a mandatory i.e., the user has to enter data into it.
- Not null Integrity constraint cannot be defined using the alter table command when the table contain rows.

Check Constraint:

Check constraint can be defined to allow only a particular range of values. When the manipulation violates this constraint, the record will be rejected. Check condition cannot contain sub queries.

b) Entity Integrity

Maintains uniqueness in a record. An entity represents a table and each row of a table represents an instance of that entity. To identify each row in a table uniquely we need to use this constraint. There are 2 entity constraints:

Unique key constraint It is used to ensure that information in the column for each record is unique, as with telephone or drivers license numbers. It prevents the duplication of value with rows of a specified column in a set of column. A column defined with the constraint can allow null value. If unique key constraint is defined in more than one column i.e., combination of column cannot be specified. Maximum combination of columns that a composite unique key can contain is 16.

Primary Key Constraint

A primary key avoids duplication of rows and does not allow null values. It can be defined on one or more columns in a table and is used to uniquely identify each row in a table. These values should never

be changed and should never be null. A table should have only one primary key. If a primary key constraint is assigned to more than one column or combination of column is said to be composite primary key, which can contain 16 columns.

c) Referential Integrity

It enforces relationship between tables. To establish parent-child relationship between 2 tables having a common column definition, we make use of this constraint. To implement this, we should define the column in the parent table as primary key and same column in the child table as foreign key referring to the corresponding parent entry.

Foreign key

A column or combination of column included in the definition of referential

integrity, which would refer to a referenced key.

Referenced key

It is a unique or primary key upon which is defined on a column belonging to the parent table.

Types of data constrains.

a) not null constraint at column level.

Syntax:<col><datatype>(size)not null

b) unique constraint

Unique constraint at column level.

Syntax:<col><datatype>(size)unique;

c) unique constraint at table level:

Syntax:Create table tablename(col=format,col=format,unique(<col1>,<col2>);

d) primary key constraint at column level

Syntax:<col><datatype>(size)primary key;

e) primary key constraint at table level.

Syntax: Create table tablename(col=format,col=format primary key(col1>,<col2>);

f) foreign key constraint at column level.

Syntax: <col><datatype>(size>) references <tablename>[<col>];

g) foreign key constraint at table level

Syntax: foreign key(<col>[,<col>])references <tablename>[(<col>,<col>)

h) check constraint:check constraint constraint at column level.

Syntax: <col><datatype>(size) check(<logical expression>)

i) check constraint constraint at table level.

Syntax: check(<logical expression>)

CONCLUSION :

Thus we study that creating a database of the same schema using Data Definition Language (DDL). Use all DDL statements (Create, Alter, Drop) with all possible options and constraints (Primary key, Foreign Key, unique, Not Null, Default, Check etc.)

Teacher Sign

PRACTICAL NO. 02

AIM : Design at least 10 SQL queries for suitable database application using SQL DML statements to retrieve, insert, delete and update data and queries which involves DML Features like Set Operation, Set Comparisons, Aggregation, group by, having, order by, nested queries.

THEORY:

DML: (select, from, where, as clauses, rename, Insert, delete, update, case construct [Write description and syntax of these commands])

DML COMMAND

DML commands are the most frequently used SQL commands and is used to query and manipulate the existing database objects. Some of the commands are Insert, Select, Update, Delete.

Insert Command

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

Select Commands

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

Update Command

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

Delete command

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

INSERT COMMAND

Inserting a single row into a table:

Syntax: insert into <table name> values (value list)

Example: insert into s values(„s3‟,‟sup3‟,‟blore‟,10)

Inserting more than one record using a single insert commands:

Syntax: insert into <table name> values (&col1, &col2, ...)

Example: Insert into stud values(&reg, "&name", &percentage);

Skipping the fields while inserting:

Insert into <table name(coln names to which data to be inserted)> values (list of values);

Other way is to give null while passing the values.

SELECT COMMANDS

Selects all rows from the table

Syntax: Select * from table_name;

Example; Select * from IT;

The retrieval of specific columns from a table:

It retrieves the specified columns from the table

Syntax: Select column_name1,..... ,column_namen from table name;

Example: Select empno, empname from emp;

Elimination of duplicates from the select clause:

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

Syntax: Select DISTINCT col1, col2 from table name;

Example: Select DISTINCT job from emp;

Select command with where clause:

To select specific rows from a table we include „where" clause in the select command. It

can appear only after the „from" clause.

Syntax: Select column_name1,..... ,column_namen from table name where condition;

Example: Select empno, empname from emp where sal>4000;

Select command with order by clause:

Syntax: Select column_name1,..... ,column_namen from table name where condition

order by colmnname;

Example: Select empno, empname from emp order by empno;

Select command to create a table:

Syntax: create table <tablename> as select * from existing_tablename;

Example: create table emp1 as select * from emp;

Select command to insert records:

Syntax: insert into tablename (select columns from existing_tablename);

Example: insert into emp1 (select * from emp);

UPDATE COMMAND

Syntax:update tablename set field=values where condition;

Example:Update emp set sal = 10000 where empno=135;

a) updating all rows

Syntax:Update <tablename> set <col>=<exp>,<col>=<exp>;

b) updating seleted records.

Syntax: Update <tablename> set <col>=<exp>,<col>=<exp>where <condition>;

SQL Commands:

DELETE COMMAND

Syntax: Delete from table where conditions;

Example:delete from emp where empno=135;

a) remove all rows

Syntax: delete from <tablename>;

b) removal of a specified row/s

Syntax: delete from <tablename> where <condition>;

SQL FEATURES: AGGREGATE FUNCTIONS

are functions that take a collection (a set or multiset) of values as input and return a single value. SQL offers five built-in aggregate functions:

- **Average**:avg
- **Minimum**:min
- **Maximum**:max

- **Total**:sum

- **Count**:count

The input to sum and avg must be a collection of numbers, but the other operators can operate on collections of nonnumeric data types, such as strings, as well.

Group by clause:

There are circumstances where we would like to apply the aggregate function not only to a single set of tuples, but also to a group of sets of tuples; we specify this wish in SQL using the group by clause. The attribute or attributes given in the group by clause are used to form groups. Tuples with the same value on all attributes in the group by clause are placed in one group.

Having Clause:

At times, it is useful to state a condition that applies to groups rather than to tuples. To express such a query, we use the having clause of SQL. SQL applies predicates in the having clause after groups have been formed, so aggregate functions may be used.

Order by clause:

The order by clause causes the tuples in the result of a query to appear in sorted order. By default, the order by clause lists items in ascending order. To specify the sort order, we may specify **desc** for descending order or **asc** for ascending order.

SQL FEATURES: SET OPERATIONS

The Set operator combines the result of 2 queries into a single result. The following are the operators:

· Union · Union all

· Intersect · Difference

The rules to which the set operators are strictly adhere to :

· The queries which are related by the set operators should have a same number of column and column definition; such query should not contain a type of long. Labels under which the result is displayed are those from the first select statement.

SQL commands:

Union: Returns all distinct rows selected by both the queries

Syntax: Query1 Union Query2;

Union all: Returns all rows selected by either query including the duplicates.

Syntax: Query1 Union all Query2;

Intersect: Returns rows selected that are common to both queries.

Syntax: Query1 Intersect Query2;

Minus: Returns all distinct rows selected by the first query and are not by the second

Syntax: Query1 minus Query2;

SQL FEATURES: JOINS

SELF Join, Equi join, non-equi join, inner join, Natural join, outer join, LEFT outer join, right outer join, full outer join, cross join ,'ON' clause [Write description]

Nested Queries: Nesting of queries one within another is known as a nested queries.

Sub queries The query within another is known as a sub query. A statement containing sub query is called parent statement. The rows returned by sub query are used by the parent statement.

Types

1. Sub queries that return several values

Sub queries can also return more than one value. Such results should be made use along with the operators in and any.

2. Multiple queries

Here more than one sub query is used. These multiple sub queries are combined by means of „and" & „or" keywords

3. Correlated sub query

A sub query is evaluated once for the entire parent statement whereas a correlated Sub query is evaluated once per row processed by the parent statement.

Relating Data through Join Concept

The purpose of a join concept is to combine data spread across tables. A join is actually performed by the „where" clause which combines specified rows of tables.

Syntax; select columns from table1, table2 where logical expression;

Types of Joins 1. Simple Join 2. Self Join 3. Outer Join 4. Inner Join

1. Simple Join

a) Equi-join: A join, which is based on equalities, is called equi-join.

b) Non Equi-join: It specifies the relationship between

Table Aliases

Table aliases are used to make multiple table queries shorted and more readable. We give an alias name to the table in the „from‟ clause and use it instead of the name throughout the query.

Self join: Joining of a table to itself is known as self-join. It joins one row in a table to another. It can compare each row of the table to itself and also with other rows of the same table.

Outer Join: It extends the result of a simple join. An outer join returns all the rows returned by simple join as well as those rows from one table that do not match any row from the table. The symbol (+) represents outer join.

Inner join: Inner join returns the matching rows from the tables that are being joined.

SQL Commands:

Nested Queries:

Example: select ename, eno, address where salary >(select salary from employee where ename =‟jones‟);

1.**Subqueries that return several values**

Example: select ename, eno, from employee where salary <any (select salary from employee where deptno =10‟);

3**.Correlated subquery**

Example: select * from emp x where x.salary > (select avg(salary) from emp where deptno =x.deptno);

Simple Join

a) Equi-join

Example: select * from item, cust where item.id=cust.id;

b) Non Equi-join

Example: select * from item, cust where item.id<cust.id;

Self join

Example: select * from emp x ,emp y where x.salary >= (select avg(salary) from x.emp where x. deptno =y.deptno);

Outer Join

Example: select ename, job, dname from emp, dept where emp.deptno (+) = dept.deptno;

CONCLUSION

Thus we study to design SQL queries for suitable database application using SQL DML statements to retrieve, insert, delete and update data and queries which involves DML Features like Set Operation, Set Comparisons, Aggregation, group by, having, order by, nested queries.

Teacher Sign

PRACTICAL NO. 03

AIM: SQL queries to demonstrate Transaction control language(TCL): commit, savepoint, Rollback and Data Control Language (DCL): Grant Revoke.

THEORY

TCL command :

Transaction Control Language(TCL) commands are used to manage transactions in database. These are used to manage the changes made by DML statements. It also allows statements to be grouped together into logical transactions.

1. Commit command

Commit command is used to permanently save any transaction into database.
Following is Commit command's syntax,

Commit ;

2. Rollback command

This command restores the database to last commited state. It is also use with savepoint command to jump to a savepoint in a transaction.
Following is Rollback command's syntax,

rollback to savepoint-name;

3. Savepoint command

savepoint command is used to temporarily save a transaction so that you can rollback to that point whenever necessary.
Following is savepoint command's syntax,

savepoint savepoint-name;

DCL command :

Data Control Language(DCL) is used to control privilege in Database. To perform any operation in the database, such as for creating tables, sequences or views we need privileges. Privileges are of two types,

System : creating session, table etc are all types of system privilege.
Object : any command or query to work on tables comes under object privilege.

DCL defines two commands,

1 Grant Command

Gives user access privileges to database.

To Allow a User to create Session

grant create session to *username*;

2 Revoke Command

Take back permissions from user.

To take back Permissions

revoke create table from *username;*

CONCLUSION

Thus we studied the Transaction control language(TCL): commit, savepoint, Rollback and Data Control Language (DCL): Grant Revoke.

Teacher Sign with Date

PRACTICAL NO. 04

AIM : SQL queries to demonstrate View, Triggers and Indexing.

THEORY :

Views:

A view is the tailored presentation of data contained in one or more table and can also be said as restricted view to the data's in the tables. A view is a "virtual table" or a "stored query" which takes the output of a query and treats it as a table. The table

upon which a view is created is called as base table. A view is a logical table based on a table or another view. A view contains no data of its own but is like a window through which data from tables can be viewed or changed. The tables on which a view is based are called base tables. The view is stored as a SELECT statement in the data dictionary
Advantages of a view:

a. Additional level of table security.
b. Hides data complexity.
c. Simplifies the usage by combining multiple tables into a single table.
d. Provides data's in different perspective.

Types of view:
Horizontal -> enforced by where cause
Vertical -> enforced by selecting the required columns

SQL Commands
Creating and dropping view:
Syntax:
Create [or replace] view <view name> [column alias names] as <query> [with <options> conditions];

Drop view <view name>;

Example:

1. Create or replace view empview as select * from emp;
2. Drop view empview;

Trigger

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

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

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

Benefits of Triggers

Triggers can be written for the following purposes:

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

Creating Triggers

The syntax for creating a trigger is:

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

Index

Indexes are used to retrieve data from the database very fast. The users cannot see the indexes, they are just used to speed up searches/queries.

Note: Updating a table with indexes takes more time than updating a table without (because the indexes also need an update). So, only create indexes on columns that will be frequently searched against.

CREATE INDEX Syntax

```
CREATE INDEX index_name

ON table_name (column1, column2, ...);
```

CONCLUSION:

Thus we studied the View, Triggers and Indexing.

Teacher Sign

PRACTICAL NO. 05

AIM : Perform DML Queries using PL/SQL.

THEORY

Describe PL/SQL block.

Following is the basic structure of a PL/SQL block:

```
DECLARE
        <declarations section>
BEGIN
        <executable command(s)>
EXCEPTION
        <exception handling>
END;
```

PL/SQL Program Units

A PL/SQL unit is any one of the following:

- PL/SQL block
- Function
- Package
- Package body
- Procedure
- Trigger
- Type
- Type body

Variable Declaration in PL/SQL

The syntax for declaring a variable is:

variable_name [CONSTANT] datatype [NOT NULL] [:= | DEFAULT initial_value]

Initializing Variables in PL/SQL

Whenever you declare a variable, PL/SQL assigns it a default value of NULL. If you want to initialize a variable with a value other than the NULL value, you can do so during the declaration, using either of the following:

- The **DEFAULT** keyword
- The **assignment** operator

For example:

```
counter binary_integer := 0;
greetings varchar2(20) DEFAULT 'Have a Good Day';
```

Variable Scope in PL/SQL

There are two types of variable scope:

- **Local variables** - Variables declared in an inner block and not accessible to outer blocks.
- **Global variables** - Variables declared in the outermost block or a package.

Assigning SQL Query Results to PL/SQL Variables

The following program assigns values from the above table to PL/SQL variables using the **SELECT INTO clause** of SQL:

```
DECLARE
c_id customers.id%type := 1;
c_name customerS.No.ame%type;
c_addr customers.address%type;
c_sal customers.salary%type;
BEGIN
SELECT name, address, salary INTO c_name, c_addr, c_sal
FROM customers
WHERE id = c_id;
dbms_output.put_line
        ('Customer ' ||c_name || ' from ' || c_addr || ' earns ' || c_sal);
END;
```

- **PL/SQL – Conditions**

1 IF-THEN Statement :

Syntax

```
IF condition THEN
        S;
END IF;
```

2 IF-THEN-ELSE Statement :

Syntax

```
IF condition THEN
        S1;
ELSE
        S2;
END IF;
```

3 IF-THEN-ELSIF Statement :

Syntax

```
IF(boolean_expression 1)THEN
        S1; -- Executes when the boolean expression 1 is true
ELSIF( boolean_expression 2) THEN
```

```
        S2; -- Executes when the boolean expression 2 is true
    ELSIF( boolean_expression 3) THEN
        S3; -- Executes when the boolean expression 3 is true
    ELSE
        S4; -- executes when the none of the above condition is true
    END IF;
```

4 CASE Statement :

Syntax

```
    CASE selector
        WHEN 'value1' THEN S1;
        WHEN 'value2' THEN S2;
        WHEN 'value3' THEN S3;
        ...
        ELSE Sn; -- default case
    END CASE;
```

5 Searched CASE Statement :

Syntax

```
    CASE
        WHEN selector = 'value1' THEN S1;
        WHEN selector = 'value2' THEN S2;
        WHEN selector = 'value3' THEN S3;
        ...
        ELSE Sn; -- default case
    END CASE;
```

6 Nested IF-THEN-ELSE Statements :

Syntax

```
    IF( boolean_expression 1)THEN
        -- executes when the boolean expression 1 is true
        IF(boolean_expression 2) THEN
            -- executes when the boolean expression 2 is true
              sequence-of-statements;
        END IF;
    ELSE
        -- executes when the boolean expression 1 is not true
          else-statements;
    END IF;
```

- **PL/SQL – Loops**

1 Basic Loop Statement

Syntax

```
LOOP
        Sequence of statements;
END LOOP;
```

2 WHILE LOOP Statement

Syntax

```
WHILE condition LOOP
        sequence_of_statements
END LOOP;
```

3 FOR LOOP Statement

Syntax

```
FOR counter IN initial_value .. final_value
LOOP
        sequence_of_statements;
END LOOP;
```

4 Reverse FOR LOOP Statement

Syntax

```
DECLARE
        a number(2) ;
BEGIN
        FOR a IN REVERSE 10 .. 20 LOOP
                dbms_output.put_line('value of a: ' || a);
        END LOOP;
END;
```

5 Nested Loops

Syntax

```
LOOP
        Sequence of statements1
        LOOP
                Sequence of statements2
        END LOOP;
END LOOP;
```

6 Labeling a PL/SQL Loop

Syntax

```
DECLARE
        i number(1);
        j number(1);
```

```
BEGIN
    << outer_loop >>
    FOR i IN 1..3 LOOP
    << inner_loop >>
        FOR j IN 1..3 LOOP
        dbms_output.put_line('i is: '|| i || ' and j is: ' || j);
    END loop inner_loop;
END loop outer_loop;
END;
```

- **PL/SQL – Cursors**

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

There are two types of cursors:

- Implicit cursors
- Explicit cursors

Declaring the Cursor

Declaring the cursor defines the cursor with a name and the associated SELECT statement.

For example:

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

- **PL/SQL – Records**

A **record** is a data structure that can hold data items of different kinds. Records consist of different fields, similar to a row of a database table.

For example, you want to keep track of your books in a library. You might want to track the following attributes about each book, such as Title, Author, Subject, Book ID. A record containing a field for each of these items allows treating a BOOK as a logical unit and allows you to organize and represent its information in a better way.
PL/SQL can handle the following types of records:

- Table-based
- Cursor-based records
- User-defined records

User-Defined Records
PL/SQL provides a user-defined record type that allows you to define the different record structures. The record type is defined as:

```
TYPE
type_name IS RECORD
```

```
        ( field_name1 datatype1 [NOT NULL] [:= DEFAULT EXPRESSION],
        field_name2 datatype2 [NOT NULL] [:= DEFAULT EXPRESSION],
        ...
        field_nameN datatypeN [NOT NULL] [:= DEFAULT EXPRESSION);
record-name type_name;
```

- **PL/SQL – DBMS Output**

The **DBMS_OUTPUT** is a built-in package that enables you to display output, debugging information, and send messages from PL/SQL blocks, subprograms, packages, and triggers.

Let us look at a small code snippet that will display all the user tables in the database. Try it in your database to list down all the table names:

```
BEGIN
        dbms_output.put_line (user || ' Tables in the database:');
        FOR t IN (SELECT table_name FROM user_tables)
        LOOP
                dbms_output.put_line(t.table_name);
        END LOOP;
END
```

Conclusion:

Hence we perform program to demonstrate PL/SQL block.

Teacher Sign

PRACTICAL NO. 06

AIM : Install MongoDB, run MongoDB on your OS and setup a python environment with MongoDB.

THEORY

MongoDB:

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

Database

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

Collection

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

Document

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

The following table shows the relationship of RDBMS terminology with MongoDB.

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

Sample Document

Following example shows the document structure of a blog site, which is simply a comma separated key value pair.

```
{
        _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,
        comments: [
                {
                        user:'user1',
                        message: 'My first comment',
                        dateCreated: new Date(2011,1,20,2,15),
                        like: 0
                },
                {
                        user:'user2',
                        message: 'My second comments',
                        dateCreated: new Date(2011,1,25,7,45),
                        like: 5
                }
            ]
}
```

_id is a 12 bytes hexadecimal number which assures the uniqueness of every document. You can provide _id while inserting the document. If you don't provide then MongoDB provides a unique id for every document. These 12 bytes first 4 bytes for the current timestamp, next 3 bytes for machine id, next 2 bytes for process id of MongoDB server and remaining 3 bytes are simple incremental VALUE.

Any relational database has a typical schema design that shows number of tables and t herelationship between these tables. While in MongoDB, there is no concept of relationship.

Advantages of MongoDB over RDBMS

- 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.
- Tuning.
- 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.

Why Use MongoDB?

-Document Oriented Storage: Data is stored in the form of JSON style documents.
- Index on any attribute
- Replication and high availability

-Auto-sharding
- Rich queries
- Fast in-place updates
- Professional support by MongoDB

Where to Use MongoDB?

- Big Data
- Content Management and Delivery
- Mobile and Social Infrastructure
- User Data Management
- Data Hub

Install MongoDB

1. sudo apt-get update
2. sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv 7F0CEB10

3. echo 'deb http://downloads-distro.mongodb.org/repo/ubuntu-upstart dist 10gen' | sudo tee /etc/apt/sources.list.d
4.

for ubuntu 12.04

echo "deb [arch=amd64] http://repo.mongodb.org/apt/ubuntu precise/mongodb-org/3.4 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-3.4.list

for ubuntu 14.04

echo "deb [arch=amd64] http://repo.mongodb.org/apt/ubuntu trusty/mongodb-org/3.4 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-3.4.list

for 16.04

echo "deb [arch=amd64,arm64] http://repo.mongodb.org/apt/ubuntu xenial/mongodb-org/3.4 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-3.4.list

4. sudo apt-get update
5. sudo apt-get install -y mongodb-org

Install Python3

1. sudo apt-get update
2. sudo apt-get install python3

In order to be able to connect to MongoDB with Python, you need to install the PyMongo driver package. To install PyMongo driver package we need pip (python3 version)
-Pip is intended to be an improved python package installer. It integrates with virtualenv, does not do partial installs, can save package states for replaying, can install from non-egg sources and

can install from version control repositories.

Install Pymongo

1. python3 --version
2 sudo apt-get install python3-pip
3 sudo pip3 install pymongo

restart Machine

Run MongoDB

- Start MongoDB
  sudo service mongodb start
- Stop MongoDB
  sudo service mongodb stop
- Restart MongoDB
  sudo service mongodb restart
- To use MongoDB
  mongo
  This will connect you to running MongoDB instance.

A quick way to test whether there is a MongoDB instance already running on your local machine is to type mongo at the command-line. This will start the MongoDB admin console, which attempts to connect to a database server running on the default port (27017).

- MongoDB Help
  To get a list of commands,
  -db.help() in MongoDB client
- MongoDB Statistics
  To get stats about MongoDB server,
  db.stats() in MongoDB client.

This will show the database name, number of collection and documents in the database.

Conclusion :

Hence we Installed MongoDB, executed MongoDB on your ubuntu and setup a python environment with MongoDB.

Teacher Sign

PRACTICAL NO. 07

AIM

Connect to MongoDB with python, get a Database Handle . Create a collection and insert a document into it.

THEORY

Data in MongoDB has a flexible schema.documents in the same collection. They 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.

Some considerations while designing Schema in MongoDB

- Design your schema according to user requirements.
- Combine objects into one document if you will use them together. Otherwise separate them (but make sure there should not be need of joins).
- Duplicate the data (but limited) because disk space is cheap as compare to compute time.
- Do joins while write, not on read.
- Optimize your schema for most frequent use cases.
- Do complex aggregation in the schema.

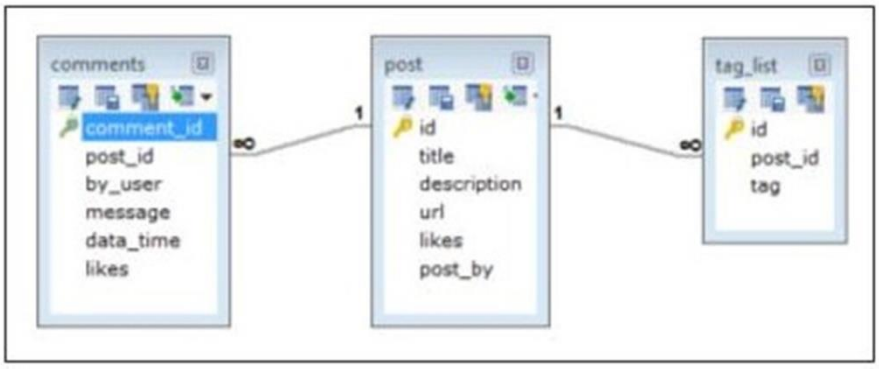
Example

Suppose a client needs a database design for his blog/website and see the differences between RDBMS and MongoDB schema design.

Website has the following requirements.

- Every post has the unique title, description and url.
- Every post can have one or more tags.
- Every post has the name of its publisher and total number of likes.
- Every post has comments given by users along with their name, message, data- time and likes.
- On each post, there can be zero or more comments.

In RDBMS schema, design for above requirements will have minimum three tables.



While in MongoDB schema, design will have one collection post and the following structure:

```
{
_id: POST_ID
title: TITLE_OF_POST,
description: POST_DESCRIPTION,
by: POST_BY,
url: URL_OF_POST,
tags: [TAG1, TAG2, TAG3],
likes: TOTAL_LIKES,
comments: [
        {
                user:'COMMENT_BY',
                message: TEXT,
                dateCreated: DATE_TIME,
                like: LIKES
        },
        {
                user:'COMMENT_BY',
                message: TEXT,
                dateCreated: DATE_TIME,
                like: LIKES
        }
    ]
}
```

So while showing the data, in RDBMS you need to join three tables and in MongoDB, data will be shown from one collection only.

Create Database:

The use Command

MongoDB use DATABASE_NAME is used to create database. The command will create a new database if it doesn't exist, otherwise it will return the existing database.

Syntax:

use DATABASE_NAME

To check your currently selected database, use the command db

>db

If you want to check your databases list, use the command show dbs.

>show dbs

In MongoDB default database is test. If you didn't create any database, then collections will be stored in test database.

Create Collection:

The createCollection() Method

MongoDB db.createCollection(name, options) is used to create collection.

Syntax:

db.createCollection(name, options)

name is name of collection to be created. Options is a document and is used to specify configuration of collection.

Following is the list of options you can use:

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

ex. >db.createCollection("mycollection")

ex.

>db.createCollection("mycol", { capped : true, autoIndexID : true, size :

6142800, max : 10000 })

Show collections

Shows all collections in db.

Syntax:

>show collections

In MongoDB, you don't need to create collection. MongoDB creates collection automatically, when you insert some document.

Syntax:

db_name.collection_name.insert(document)

ex.

db.collection1.insert("author": "abc")

this command automatically creates a collection named 'collection1'.

MongoDB – Datatypes:

MongoDB supports many datatypes. Some of them are:

- String: This is the most commonly used datatype to store the data. String in MongoDB must be UTF-
  8 valid.
- Integer: This type is used to store a numerical value. Integer can be 32 bit or 64 bit depending upon
  your server.
- Boolean: This type is used to store a boolean (true/ false) value.
- Double: This type is used to store floating point values.
- Min/Max Keys: This type is used to compare a value against the lowest and highest BSON elements.
- Arrays: This type is used to store arrays or list or multiple values into one key.
- Timestamp: ctimestamp. This can be handy for recording when a document has been modified or
  added.
- Object: This datatype is used for embedded documents.
- Null: This type is used to store a Null value.
- Symbol: This datatype is used identically to a string; however, it's generally reserved for languages
  that use a specific symbol type.

- Date: This datatype is used to store the current date or time in UNIX time format. You can specify
your own date time by creating object of Date and passing day, month, year into it.
- Object ID: This datatype is used to store the document's ID.
- Binary data: This datatype is used to store binary data.
- Code: This datatype is used to store JavaScript code into the document.
- Regular expression: This datatype is used to store regular expression.

Documents

Data in MongoDB is represented (and stored) using JSON-style documents. In PyMongo we use dictionaries to represent documents. As an example, the following dictionary might be used to represent a blog post:

```
>>> import datetime
>>> post = {"author": "Mike",
...         "text": "My first blog post!",
...         "tags": ["mongodb", "python", "pymongo"],
...         "date": datetime.datetime.utcnow()}
```

Insert Document:

To insert data into MongoDB collection, you need to use MongoDB's insert() or save()method.

The insert() Method

syntax:

```
>db.COLLECTION_NAME.insert(document)
```

The save() Method

syntax:

```
>db.COLLECTION_NAME.save(document)
```

Connect to MongoDB with python, get a Database Handle

- start mongodb one terminal

$mongo

- start python3 on other terminal (install pymongo before connection)

$python3

>>>import pymongo

- To make a Connection with MongoClient:

The first step when working with PyMongo is to create a MongoClient to the running mongo instance.

>>> from pymongo import MongoClient

>>> client = MongoClient()

- Getting a Database/ database Handle.

A single instance of MongoDB can support multiple independent databases. When working with PyMongo you access databases using attribute style access on MongoClient instances:

>>> db = client.database_name

If your database name is such that using attribute style access won't work (like database-name), you can use dictionary style access instead:

>>> db = client['database-name']

Getting a Collection

A collection is a group of documents stored in MongoDB, and can be thought of as roughly the equivalent of a table in a relational database. Getting a collection in PyMongo works the same as getting a database:

>>> collection = db.collection_name

or (using dictionary style access):

>>> collection = db['test-collection']

If collection does not exist in database it creates a collection.

Collections (and databases) in MongoDB is that they are created lazily - none of the above

commands have actually performed any operations on the MongoDB server. Collections and databases are created when the first document is inserted into them.

Insert a document using python

To insert a document into a collection we can use the insert_one() or insert_many() method:

```
>>> post1 = db.posts                    // creates acollections named posts and assign it as variable                    //post1
>>> post_id = post1.insert_one(post).inserted_id    // insert a document in posts collection and returns                    //_id of inserted document
>>> post_id
```

When a document is inserted a special key, "_id" , is automatically added if the document doesn't already contain an "_id" key. The value of "_id" must be unique across the collection. insert_one() returns an instance of InsertOneResult .

After inserting the first document, the posts collection has actually been created on the server. We can verify this by listing all of the collections in our database:

```
>>>db.collection_names()
```

Bulk Inserts

- create a list of documents to be inserted
- pass that list as the first argument to insert_many(list_name)

```
>>> result = posts.insert_many(new_posts)
```

CONCLUSION

Hence we Connect to MongoDB with python and Create a collection and insert a document into it using python.

Teacher Sign

PRACTICAL NO. 08

AIM To perform operations on collection.

a) Retrieve all documents in a collection which matches certain property.
b) Perform queries that uses MongoDB query operators.

THEORY

MongoDB:

The find() Method

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

Syntax

>db.COLLECTION_NAME.find()

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

The pretty() Method

Display the results in a formatted way, you can use pretty() method.

Syntax

>db.mycol.find().pretty()

RDBMS Where Clause Equivalents in MongoDB

Operation	Syntax	Example	RDBMS Equivalent
Equality	{<key>:<value>}	db.mycol.find({"by":"tutorials point"}).pretty()	where by = 'tutorials point'
Less Than	{<key>:{$lt:<value>}}	db.mycol.find({"likes":{$lt:50}}).pretty()	where likes < 50
Less Than Equals	{<key>:{$lte:<value>}}	db.mycol.find({"likes":{$lte:50}}).pretty()	where likes <= 50
Greater Than	{<key>:{$gt:<value>}}	db.mycol.find({"likes":{$gt:50}}).pretty()	where likes > 50
Greater Than Equals	{<key>:{$gte:<value>}}	db.mycol.find({"likes":{$gte:50}}).pretty()	where likes >= 50
Not Equals	{<key>:{$ne:<value>}}	db.mycol.find({"likes":{$ne:50}}).pretty()	where likes != 50

To query the document on the basis of some condition, you can use operations in table 1:

AND in MongoDB

Syntax

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

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

OR in MongoDB

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

Syntax

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

retrieve documents using python:

find_one()

to retrieve 1st document in collection'

>>>posts.find_one()

find()

To get more than a single document as the result of a query we use the find() method. Find() returns a Cursor instance, which allows us to iterate over all matching documents.

Syntax

>>>db.collection.find(query,projection) //query and projection both are optional

query is in document format.

Projection format: {field: value,field:value..} fields which we want in result.
Value is 1 or true if we want that field in result otherwise 0 or false.

For example, we can iterate over every document in the posts collection:
for p in collection.find():pprint.pprint(p)
where p refers to cursor instance

>>> for post in posts.find(): pprint.pprint(post)

To find selected documents which Matches certain property, pass that condition to find().

ex.

>>> for post in posts.find({"author": "Mike"}):pprint.pprint(post)

CONCLUSION

Hence performed operations on collections.

Teacher Sign

PRACTICAL NO. 9

AIM Perform update and delete operations on a collection

THEORY

Update document

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

>db.collection_name.update({selection_criteria},{update_modifier:{field:value,field:value...}}

ex.

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

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})

update using python

syntax:

>db.collection_name.update({selection_criteria},{“$set”:{field:value,field:value ...}}

MongoDB Save() Method

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

Syntax

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

Folloing are the MongoDB update modifiers

Modifier	Meaning	Example
$inc	Atomic Increment	"$inc":{"score":1}
$set	Set Property Value	"$set":{"username":"niall"}
$unset	Unset (delete) Property	"$unset":{"username":1}
$push	Atomic Array Append (atom)	"$push":{"emails":"foo@example.com"}
$pushAll	Atomic Array Append (list)	"$pushall":{"emails":["foo@example.com","foo2@example.com"]}
$addToSet	Atomic Append-If-Not-Present	"$addToSet":{"emails":"foo@example.com"}
$pop	Atomic Array Tail Remove	"$pop":{"emails":1}
$pull	Atomic Conditional Array Item Removal	"$pull":{"emails":"foo@example.com"}
$pullAll	Atomic Array Multi Item Removal	"$pullAll":{"emails":["foo@example.com", "foo2@example.com"]}
$rename	Atomic Property Rename	"$rename":{"emails":"old_emails"}

Delete Document

The remove() Method

MongoDB's remove() method is used to remove a document from the collection. remove() method accepts two parameters. One is deletion criteria and second is justOne flag.

- deletion criteria: (Optional) deletion criteria according to documents will be removed.
- justOne: (Optional) if set to true or 1, then remove only one document.

Syntax

>db.COLLECTION_NAME.remove(DELLETION_CRITTERIA)

Remove Only One

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

>db.COLLECTION_NAME.remove(DELETION_CRITERIA,1)

Remove All Documents

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

>db.mycol.remove()

CONCLUSION

Hence we studied updation and deletion of document in mongoDB

Teacher Sign

PRACTICAL NO. 10

AIM Perform Aggregation function on collection.

THEORY

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 groupby is an equivalent of mongodb aggregation.

The aggregate() Method

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

Syntax

>db.COLLECTION_NAME.aggregate([{$group : {_id : "grouping_field", aggrigation_field_name : {$aggregation_operation :{"$aggregation_property"}}}])

ex:

> db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$sum : 1}}}])

this counts number of documents in each group.

Expression	Description	Example
$sum	Sums up the defined value from all documents in the collection.	db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$sum : "$likes"}}}])
$avg	Calculates the average of all given values from all documents in the collection.	db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$avg : "$likes"}}}])
$min	Gets the minimum of the corresponding values from all documents in the collection.	db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$min : "$likes"}}}])
$max	Gets the maximum of the corresponding values from all documents in the collection.	db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$max : "$likes"}}}])
$push	Inserts the value to an array in the resulting document.	db.mycol.aggregate([{$group : {_id : "$by_user", url : {$push: "$url"}}}])
$addToSet	Inserts the value to an array in the resulting document but does not create duplicates.	db.mycol.aggregate([{$group : {_id : "$by_user", url : {$addToSet : "$url"}}}])
$first	Gets the first document from the source documents according to the grouping. Typically this makes only sense together with some previously applied "$sort"-stage.	db.mycol.aggregate([{$group : {_id : "$by_user", first_url : {$first : "$url"}}}])
$last	Gets the last document from the source documents according to the grouping. Typically this makes only sense together with some previously applied "$sort"-stage.	db.mycol.aggregate([{$group : {_id : "$by_user", last_url : {$last : "$url"}}}])

CONCLUSION

Hence Perform Aggregation function on collection

Teacher Sign