

MES College of Engineering Pune-01**Department of Computer Engineering**

| | |
|-----------------------------|---------------------------------|
| Name of Student: | Class: |
| Semester/Year: | Roll No: |
| Date of Performance: | Date of Submission: |
| Examined By: | Experiment No: Part A-02 |

PART: A) ASSIGNMENT NO: 02**AIM: SQL Queries:**

- Design and Develop SQL DDL statements which demonstrate the use of SQL objects such as Table, View, Index, Sequence, Synonym, different constraints etc.
- Write at least 10 SQL queries on the suitable database application using SQL DML statements.

Note: Instructor will design the queries which demonstrate the use of concepts like Insert, Select, Update, Delete with operators, functions, and set operator etc.

OBJECTIVES:

- To develop basic, intermediate and advanced Database programming skills.
- To develop basic Database administration skill.

APPRATUS:

- Operating System recommended: 64-bit Open source Linux or its derivative.
- Front End: Java/PHP/Python.
- Back End: MySQL/ Oracle Database.

THEORY:**MySQL Introduction**

- MySQL is an open source, fast, flexible, reliable, RDBMS being used for many small and big businesses.
- MySQL uses a standard form of the well-known SQL data language.
- MySQL is written in C, C++ and works on many operating systems and with many languages including PHP, PERL, C, C++, JAVA, etc.
- MySQL works very quickly and can handle large data sets.

- MySQL Server works in client/server or embedded systems.
- MySQL supports large databases, up to 50 million rows or more in a table. The default file size limit for a table is 4GB, but you can increase it to 8 million terabytes (TB).

1. MySQL Basic Commands

- **To Start MySQL**

```
#mysql -u username -p
```

Enter password:

- **To Access user on Client**

```
#mysql -h Host IP -P 3306 -u username -p
```

Enter password:

- **To Exit MySQL**

```
mysql>Exit; OR #Quit;
```

- **To check version of MySQL**

```
mysql>select version();
```

- **To check current date/time**

```
mysql>select current_date;
```

```
mysql>select now();
```

2. Create, Use, Display and remove a Database

- **To Create a Database**

```
mysql>create database [if not exists] database_name;
```

- **To Use a Database**

```
mysql>use database_name;
```

- **Displaying Databases**

SHOW DATABASES to list all the existing databases in the server.

```
mysql>show databases;
```

```
+-----+
| Database      |
+-----+
| information_schema |
| mysql          |
| performance_schema |
| test           |
.....
```

The databases "mysql", "information_schema" and "performance_schema" are system databases used internally by MySQL. A "test" database is provided during installation for your testing.

- **Removing Databases**

```
mysql> drop database [if exists] database_name;
```

3. MySQL Data Type

- Numeric Data Types
- String Data Types
- Date and Time Data Types

4. Creation of a Table

A table in MySQL is called a “RELATION” and consists of rows and columns addressed as “TUPLES” and “ATTRIBUTES” of the table respectively.

The number of tuples is called “CARDINALITY” and the number of attributes is referred to as “DEGREE” of a relation.

- **Simple Table Creation**

```
mysql> create table table_name (
        <column_name> <data_type>[(size)] ,
        <column_name> <data_type>[(size)] );
```

- **Creation of Table Using SQL Constraints**

```
mysql> Create table table_name (
        <column_name> <data_type>[(size)] <constraint> ,
        <column_name> <data_type>[(size)] <constraint> );
```

The various constraints that can be issued are:-

- NOT NULL: - Ensures that a column cannot have null values.
- DEFAULT: - Provides a default value for a column when none is specified.
- UNIQUE: - Ensures that all values in a column are different.
- CHECK: - Make sure all values in a column satisfy certain criteria.
- Primary Key: - Used to uniquely identify a row in a table.
- Foreign Key: - Used to ensure referential integrity of the data.

- **To check which table exist in current database**

```
mysql> show tables;
```

- **To view/display table structure**

```
mysql> describe table_name;
```

- **To Drop table**

```
mysql> drop table table_name;
```

- **Loading data from text file (*.txt) into table**

```
mysql>LOAD DATA LOCAL INFILE "Path and Filename" INTO TABLE  
Table_name;
```

5. Insert Value in Table

```
mysql>INSERT INTO table_name VALUES (value1,value2,value3,...);
```

OR

```
mysql>INSERT INTO table_name (column1,column2,column3,...)  
VALUES (value1, value2, value3,...);
```

6. Retrieving Data from an Existing Table

The Select statement is used to retrieve data from an existing table. There are two type of select statements that can be used:

- **Select Statement Without “Where” Clause**

```
mysql>Select <what_to_select> from <table_name>;
```

- **Select Statement With “Where” Clause**

```
mysql>Select <what_to_select> from <table_name> WHERE constraint;
```

Example: **Select * from pet WHERE pet_name='TOMMY'**; will retrieve all the tuples from the table named “pet” where the value under the field “pet_name” is “TOMMY”.

7. Update Value in table

```
mysql>UPDATE table_name  
SET field1=new-value1, field2=new-value2  
[WHERE Clause]
```

8. Delete Value from table

```
mysql>DELETE FROM table_name [WHERE Clause]
```

9. Alter Table

- **To add a field**

```
mysql>ALTER TABLE table_name  
ADD new_column_name  
[ FIRST | AFTER column_name ];
```

- **To modify the data type of a field**

```
mysql>ALTER TABLE table_name  
MODIFY column_name <new-data-type>;
```

- **To delete a field**

```
mysql>ALTER TABLE table_name  
DROP column_name;
```

- **To set a common value for a field(To set Default value)**

```
mysql>ALTER TABLE table_name ALTER Column_name SET DEFAULT value;
```

- **To change the name of a field**

```
mysql>ALTER TABLE table_name  
CHANGE <old_Column_name> <new_column_name> <data-type> ;
```

- **To change the name of a table**

```
mysql>ALTER TABLE old_table_name  
RENAME TO <new_table_name>;
```

10. MySQL-View

- **Creating View**

```
#CREATE OR REPLACE  
[ALGORITHM = {MERGE | TEMPTABLE | UNDEFINED}] VIEW  
[database_name].[view_name]  
AS [SELECT statement]  
  
OR  
#CREATE VIEW view_name AS  
SELECT column_name(s)  
FROM table_name  
WHERE condition
```

11. MySQL Index

- **Creating A Mysql Index - New Table**

If you are creating a new MySQL table you can specify a column to index by using the INDEX term as we have below.

```
#CREATE TABLE table_name(col_name1,Col_name2, INDEX (col_name));
```

We have created two fields: name and employee ID (index).

```
#CREATE TABLE employee_records ( name VARCHAR(50), employeeID INT,  
INDEX (employeeID));
```

- **Creating A Mysql Index - Existing Table**

You can also add an index to an older table that you think would benefit from some indexing. The syntax is very similar to creating an index in a new table. First, let's create the table.

```
#CREATE INDEX [index name] ON [table name]([column name]);
```

Example:

```
#CREATE TABLE employee_records2 (name VARCHAR(50), employeeID INT);  
#CREATE INDEX id_index ON employee_records2(employeeID)
```

- **Unique Index**

```
CREATE UNIQUE INDEX index_name ON table_name ( column1, column2,...);
```

```
CREATE UNIQUE INDEX AUTHOR_INDEX ON tutorials_tbl (tutorial_author)
```

```
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:

- Primary Key

```
#ALTER TABLE tbl_name ADD PRIMARY KEY (column_list):
```

- Unique Index

```
#ALTER TABLE tbl_name ADD UNIQUE index_name (col_list):
```

- Simple Index

```
#ALTER TABLE tbl_name ADD INDEX index_name (col_list):
```

- FULLTEXT index that is used for text-searching purposes

```
#ALTER TABLE tbl_name ADD FULLTEXT index_name (col_list):
```

- **Displaying INDEX Information:**

```
SHOW INDEX FROM table_name\G
```

IMPLEMENTATION:

Consider following relation and solve the queries: Create different tables given below with appropriate constraints like primary key, foreign key, check constraints, not null etc.

Account (**Acc_no**, **branch_name**, balance)

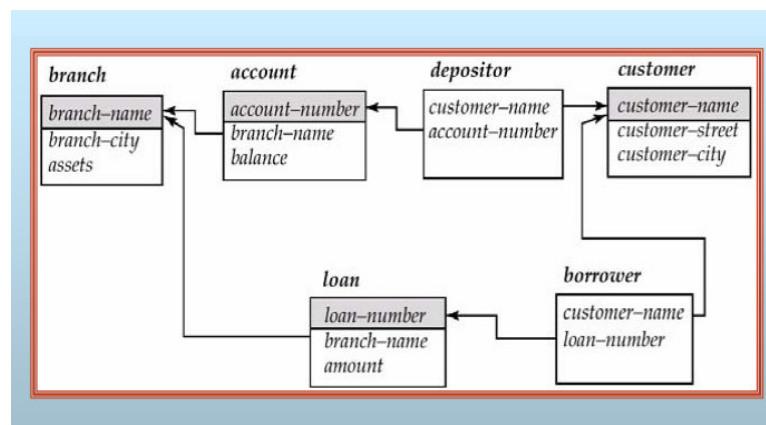
Branch (**branch_name**, branch_city, assets)

Customer (**cust_name**, cust_street, cust_city)

Depositor (**cust_name**, **acc_no**)

Loan (**loan_no**, **branch_name**, amount)

Borrower (**cust_name**, **loan_no**)



1. Find the names of all branches in loan relation.
2. Find all loan numbers for loans made at Akurdi Branch with loan amount > 12000.
3. Find all customers who have a loan from bank. Find their names, loan_no and loan amount.
4. List all customers in alphabetical order who have loan from Akurdi branch.
5. Find all customers who have an account or loan or both at bank.
6. Find all customers who have both account and loan at bank.
7. Find all customers who have account but no loan at the bank.
8. Find the average account balance at each branch
9. Find no. of depositors at each branch.
10. Find name of Customer and city where customer name starts with Letter P.
11. Display distinct cities of branch.
12. Find the branches where average account balance > 12000
13. Find number of tuples in customer relation.
14. Calculate total loan amount given by bank.
15. Delete all loans with loan amount between 1300 and 1500.
16. Delete all tuples at every branch located in Nigdi.

CONCLUSION:

QUESTIONS:

1. What are different types of databases? Explain any one (open-source) database.
2. What are DDL, DML, DCL, and TCL Languages?
3. What are primary key, unique key and foreign key?
4. How we can make use of Create statement to create multiple objects?
5. What is View? How is can helpful to user?
6. What is an Index? What are the types of indexes?
7. What is Sequence? How it is generated in MySQL?
8. What are different the Query Optimization technique's?
9. How to create synonyms in MySQL?
10. Which are the different commands used to modify database object?
11. List down the different operators that support MySQL
12. What is difference between Delete, Drop and Truncate?

MES College of Engineering Pune-01**Department of Computer Engineering**

| | |
|-----------------------------|---------------------------------|
| Name of Student: | Class: |
| Semester/Year: | Roll No: |
| Date of Performance: | Date of Submission: |
| Examined By: | Experiment No: Part A-03 |

PART: A) ASSIGNMENT NO: 03**AIM: SQL Queries – all types of Join, Sub-Query and View:**

Write at least 10 SQL queries for suitable database application using SQL DML statements.

Note: Instructor will design the queries which demonstrate the use of concepts like all types of Join, Sub-Query and View.

OBJECTIVES:

- To develop basic, intermediate and advanced Database programming skills.
- To develop basic Database administration skill.

APPARATUS:

- Operating System recommended: 64-bit Open source Linux or its derivative.
- Front End: Java/PHP/Python.
- Back End: MySQL/ Oracle Database.

THEORY:**(A). MySQL: Joins**

MySQL **JOINS** are used to retrieve data from multiple tables. A MySQL JOIN is performed whenever two or more tables are joined in a SQL statement.

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.

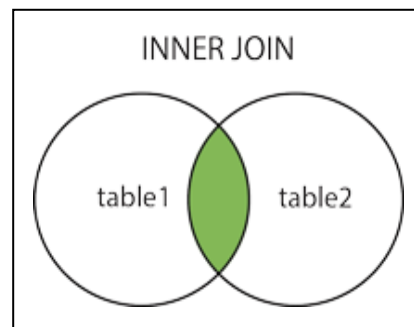
There are different types of MySQL joins:

- ✓ MySQL INNER JOIN (or sometimes called simple join)
- ✓ MySQL LEFT OUTER JOIN (or sometimes called LEFT JOIN)
- ✓ MySQL RIGHT OUTER JOIN (or sometimes called RIGHT JOIN)
- ✓ MySQL FULL JOIN (Combine Left & Right Join)

1. MySQL Inner Join

- The INNER JOIN keyword selects all rows from both tables as long as there is a match between the columns in both tables.
- **Note:** The INNER JOIN keyword selects all rows from both tables as long as there is a match between the columns.

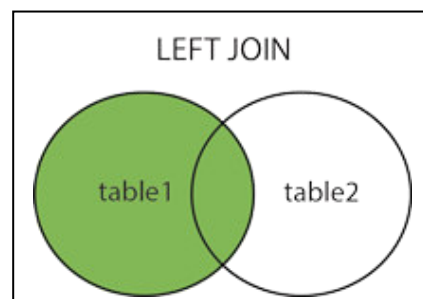
```
#SELECT column_name(s)
FROM table1
INNER JOIN table2
ON table1.column_name=table2.column_name;
OR
SELECT column_list
FROM table1
INNER JOIN table2 ON join_condition1
INNER JOIN table3 ON join_condition2
...
WHERE where_conditions;
```



2. MySQL Left Join

- The LEFT JOIN keyword returns all rows from the left table (table1), with the matching rows in the right table (table2).
- The result is NULL in the right side when there is no match.
- **Note:** The LEFT JOIN keyword returns all the rows from the left table , even if there are no matches in the right table.

```
#SELECT column_name(s)
FROM table1
LEFT JOIN table2
ON table1.column_name=table2.column_name;
OR
#SELECT column_name(s)
FROM table1
LEFT OUTER JOIN table2
ON table1.column_name=table2.column_name;
```

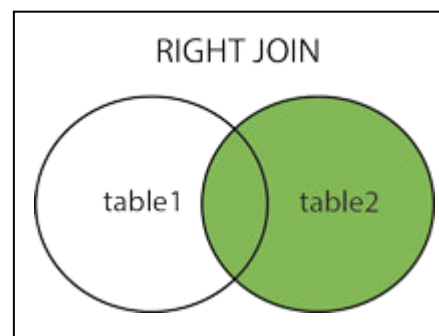


3. MySQL Right Join

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

- **Note:** The RIGHT JOIN keyword returns all the rows from the right table, even if there are no matches in the left table.

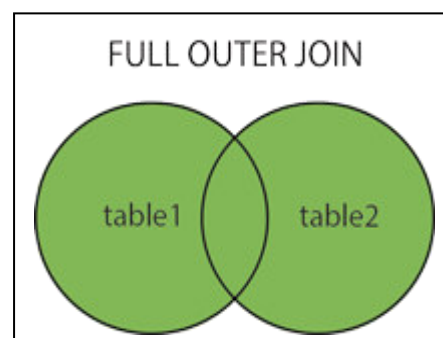
```
#SELECT column_name(s)
FROM table1
RIGHT JOIN table2
ON table1.column_name=table2.column_name;
OR
#SELECT column_name(s)
FROM table1
RIGHT OUTER JOIN table2
ON table1.column_name=table2.column_name;
```



4. MySQL Full Join

- The FULL OUTER JOIN keyword returns all rows from the left table (table1) and from the right table (table2).
- The FULL OUTER JOIN keyword combines the result of both LEFT and RIGHT joins.
- UNION Keyword can be used to combine result.
- **Note:** The FULL OUTER JOIN keyword returns all the rows from the left table (Table1), and all the rows from the right table (Table2). If there are rows in "Table1" that do not have matches in "Table2", or if there are rows in "Table2" that do not have matches in "Table1", those rows will be listed as well.

```
#SELECT column_name(s)
FROM table1
RIGHT JOIN table2
ON table1.column_name=table2.column_name
UNION
#SELECT column_name(s)
FROM table1
LEFT JOIN table2
ON table1.column_name=table2.column_name;
```

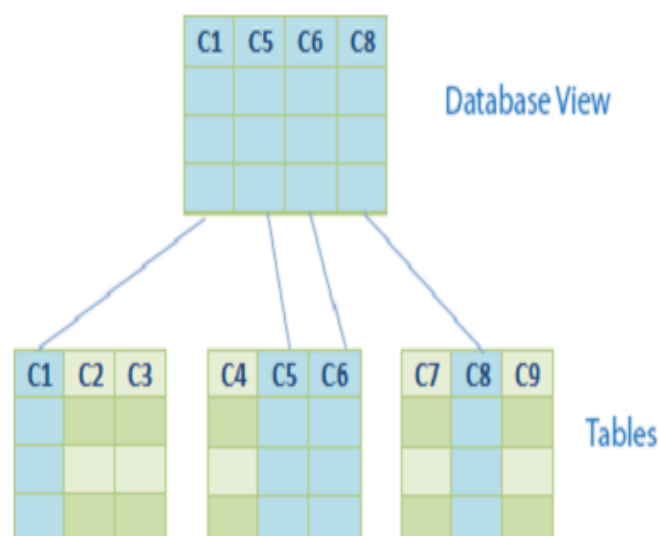


(B). MySQL-Views

- A database view is a virtual table or logical table which is defined as a SQL SELECT query with joins. Because a database view is similar to a database table, which consists of rows and columns, so you can query data against it. Most database management systems,

including MySQL, allow you to update data in the underlying tables through the database view with some prerequisites.

- A database view is dynamic because it is not related to the physical schema. The database system stores database views as a SQL SELECT statement with joins. When the data of the tables changes, the view reflects that changes as well.
- The difference between a view and a table is that views are definitions built on top of other tables (or views).
- A view can be built on top of a single or multiple tables.
- View is a data object which does not contain any data. Contents of the view are the resultant of a base table. They are operated just like base table but they don't contain any data of their own.
- MySQL supports database views or views since version 5.X. In MySQL, almost features of views conform to the SQL: 2003 standard. MySQL process queries to the views in two ways:
 - ✓ MySQL creates a temporary table based on the view definition statement and then executes the incoming query on this temporary table.
 - ✓ First, MySQL combines the incoming query with the query defined the view into one query. Then, MySQL executes the combined query.
- MySQL supports version system for views. Each time when the view is altered or replaced, a copy of the existing view is back up in arc (archive) folder which resides in a specific database folder. The name of back up file is view_name.frm-00001. If you then change the view again, MySQL will create a new backup file named view_name.frm-00002.



1. Creating View

```
#CREATE OR REPLACE
[ALGORITHM = {MERGE | TEMPTABLE | UNDEFINED}] VIEW
[database_name].[view_name]
AS [SELECT statement]

OR

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

2. Creating Updateable Views

In MySQL, views are not only read-only but also updateable. However in order to create an updateable view, the SELECT statement that defines the view has to follow several following rules:

- ✓ The SELECT statement must only refer to one database table.
- ✓ The SELECT statement must not use GROUP BY or HAVING clause.
- ✓ The SELECT statement must not use DISTINCT in the column list of the SELECT clause.
- ✓ The SELECT statement must not refer to read-only views.
- ✓ The SELECT statement must not contain any expression (aggregates, functions, computed columns...)

When you create updateable views, make sure that you follow the rules above.

```
#UPDATE view_name
SET field1=new -value1, field2=new -value2 [WHERE Clause]
```

3. Drop a View

A pre-existing view may be deleted from a database using the following statement:

```
#DROP VIEW view_name;
#DROP VIEW [IF EXISTS] [database_name].[view_name];
```

4. Getting Information About a View

All views are the result of an underlying SELECT statement. Sometimes it can be useful to find out what the SELECT statement behind a view looks like. This information can be obtained using the following:

```
#SHOW CREATE VIEW view_name;
#SHOW CREATE VIEW [database_name].[view_name];
```

5. Display a View

We can select the data from the view by executing following query:

```
#SELECT * FROM view_name;
```

6. Replacing a View

An existing view may be replaced with a new view using the same name via the CREATE OR REPLACE VIEW statement:

```
#CREATE OR REPLACE VIEW view name AS select statement
```

7. Create View With Where

```
#CREATE VIEW view_name AS  
SELECT Column_name FROM Table_name WHERE condition;
```

8. Create View With AND And OR

```
#CREATE VIEW view_name AS  
SELECT Column_name FROM Table_name WHERE (Condition1 AND  
Condition2)  
OR (Condition3 AND Condition4);
```

9. Create View With Like

```
#CREATE VIEW view_name AS SELECT Column_name FROM Table_name  
WHERE Column_name NOT LIKE "Pattern%"  
AND Column_name NOT LIKE"% Pattern";
```

10. Create View With Group By

```
#CREATE VIEW view_name AS SELECT Column_name FROM Table_name  
GROUP BY Column_name;
```

(C). Sub Queries:

- A MySQL subquery is a query that is nested inside another query such as SELECT, INSERT, UPDATE or DELETE.
- A MySQL subquery is also can be nested inside another subquery.
- A MySQL subquery is also called an inner query, while the query that contains the subquery is called an outer query.
- **Sub-queries: Guidelines**
 - ✓ A subquery must be enclosed in parentheses.
 - ✓ Use single-row operators with single-row subqueries, and use multiple-row operators with multiple-row subqueries.
 - ✓ If a subquery (inner query) returns a null value to the outer query, the outer query will not return any rows when using certain comparison operators in a WHERE clause.

| | |
|--------|----------------------|
| Select | <i>select_list</i> |
| From | <i>table</i> |
| Where | <i>expr operator</i> |

(Select *select_list*
From *table*);

- **Example:**

Let's take a look at the following subquery that returns employees who locate in the offices in the USA.

- ✓ The subquery returns all *offices codes* of the offices that locate in the USA.
- ✓ The outer query selects the last name and first name of employees whose office code is in the result set returned from the subquery.

| | |
|--|---|
| Outer Query | Subquery or Inner Query |
| ↓ | ↓ |
| <pre>SELECT lastname, firstname FROM employees WHERE officeCode IN</pre> | <pre>(SELECT officeCode FROM offices WHERE country = 'USA')</pre> |

IMPLEMENTATION

1. Consider following relation and solve the queries: Create different tables given below with appropriate constraints like primary key, foreign key, check constraints, not null etc.

Account (Acc_no, branch_name, balance)

Branch (branch_name, branch_city, assets)

Customer (cust_name, cust_street, cust_city)

Depositor (cust_name, acc_no)

Loan (loan_no, branch_name, amount)

Borrower (cust_name, loan_no)

1. Create a View1 to display List all customers in alphabetical order who have loan from Pune_Station branch.
2. Create View2 on branch table by selecting any two columns and perform insert update delete operations.
3. Create View3 on borrower and depositor table by selecting any one column from each table perform insert update delete operations.
4. Create Union of left and right joint for all customers who have an account or loan or both at bank

5. Display content of View1, View2, View3
6. Create Simple and Unique index.
7. Display index Information
8. Truncate table Customer.

2. Consider following Relation:

Companies (comp_id, name, cost, year)

| | | | |
|------|------|------|------|
| C001 | ONGC | 2000 | 2010 |
| C002 | HPCL | 2500 | 2012 |
| C005 | IOCL | 1000 | 2014 |
| C006 | BHEL | 3000 | 2015 |

Orders (comp_id, domain, quantity)

| | | |
|------|---------|-----|
| C001 | Oil | 109 |
| C002 | Gas | 121 |
| C005 | Telecom | 115 |

Create above tables with appropriate constraints execute the following query:

1. Find names, costs, domains and quantities for companies using inner join.
2. Find names, costs, domains and quantities for companies using left outer join.
3. Find names, costs, domains and quantities for companies using right outer join.
4. Find names, costs, domains and quantities for companies using Union operator.
5. Create View View1 by selecting both tables to show company name and quantities.
6. Create View2 on branch table by selecting any two columns and perform insert update delete operations.
7. Display content of View1, View2.

CONCLUSION:

QUESTIONS:

1. What is joins? How to optimize joins in MySQL?
2. Difference between inner join and outer join.
3. How many types of JOIN are supported by MySQL? Which are they? Explain
4. What are advantages and disadvantages of database view?
5. What is use of sub-queries? Explain.

MES College of Engineering Pune-01**Department of Computer Engineering**

| | |
|-----------------------------|--|
| Name of Student: | Class: |
| Semester/Year: | Roll No: |
| Date of Performance: | Date of Submission: |
| Examined By: | Experiment No: Part A-04 & A-05 |

PART: A) ASSIGNMENT NO: 04, 05

AIM: Unnamed PL/SQLcode block: Use of Control structure and Exception handling is mandatory.

OBJECTIVES:

- To develop basic, intermediate and advanced Database programming skills.
- To learn the concept of procedural language.
- To learn Control structure and Exception handling.

APPRATUS:

- Operating System recommended: 64-bit Open source Linux or its derivative
- Database: MySQL/ Oracle 11g Database.

THEORY:**PL/SQL Introduction:**

- PL SQL basically stands for "Procedural Language extensions to SQL".
- It is Extension of Structured Query Language (SQL) that is used in Oracle.
- Unlike SQL, PL/SQL allows the programmer to write code in procedural format.
- It combines the data manipulation power of SQL with the processing power of procedural language to create a super powerful SQL queries.
- It allows the programmers to instruct the compiler 'what to do' through SQL and 'how to do' through its procedural way.
- PL/SQL is a completely portable, high-performance transaction-processing language.
- PL/SQL provides a built-in, interpreted and OS independent programming environment.

| | |
|--|--------------------|
| DECLARE Variable declaration | (Optional) |
| BEGIN Program Execution SQL Statement | (Mandatory) |
| EXCEPTION Exception handling | (Optional) |
| END; | (Mandatory) |

A. Control Statements IF:

- PL/SQL supports the programming language features like conditional statements and iterative statements.
- Its programming constructs are similar to how you use in programming languages like Java and C++.
- There are different syntaxes for the IF-THEN-ELSE statement.

Syntax: (IF-THEN statement):

IF condition

THENStatement: It **is** executed **when** condition **is true**}**END IF;****Syntax: (IF-THEN-ELSE statement):**

IF condition

THEN{statements **to execute when** condition **is TRUE**...}**ELSE**{statements **to execute when** condition **is FALSE**...}**END IF;****Syntax: (IF-THEN-ELSIF statement):**

IF condition1

THEN{statements **to execute when** condition1 **is TRUE**...}

ELSIF condition2

```

        THEN
            {statements to execute when condition2 is TRUE...}
    END IF;

```

Syntax: (IF-THEN-ELSIF-ELSE statement):

```

IF condition1
    THEN
        {statements to execute when condition1 is TRUE...}
    ELSIF condition2
    THEN
        {statements to execute when condition2 is TRUE...}
    ELSE
        {statements to execute when both condition1 and condition2 are FALSE...}
    END IF;

```

B. PL/SQL Loop:

- The PL/SQL loops are used to repeat the execution of one or more statements for specified number of times.
- These are also known as iterative control statements.
- **Syntax for a basic loop:**

```

    LOOP
        Sequence of statements;
    END LOOP;

```

- **Types of PL/SQL Loops**

1. Basic Loop / Exit Loop

```

    LOOP
        statements;
    EXIT;
        {or EXIT WHEN condition;}
    END LOOP;

```

2. While Loop

```

    WHILE <condition>
        LOOP statements;
    END LOOP;

```

Important steps to follow when executing a while loop:

- ✓ Initialise a variable before the loop body.
- ✓ Increment the variable in the loop.

- ✓ EXIT WHEN statement and EXIT statements can be used in while loops but it's not done oftenly.

3. For Loop

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

- ✓ initial_value : Start integer value
- ✓ final_value : End integer value

Important steps to follow when executing a while loop:

- ✓ The counter variable is implicitly declared in the declaration section, so it's not necessary to declare it explicitly.
- ✓ The counter variable is incremented by 1 and does not need to be incremented explicitly.
- ✓ EXIT WHEN statement and EXIT statements can be used in FOR loops but it's not done oftenly.

4. GOTO Statement

```
BEGIN
    ...
    GOTO insert_row;
    ...
    <<insert_row>>
    INSERT INTO emp VALUES ...
END;
```

C. Exception Handling

- An exception is an error condition during a program execution. PL/SQL supports programmers to catch such conditions using EXCEPTION block in the program and an appropriate action is taken against the error condition. There are two types of exceptions.

1. System-defined exceptions

2. User-defined exceptions

DECLARE

<declarations section>

BEGIN

<executable command(s)>

EXCEPTION

<exception handling goes here >

WHEN exception1 THEN

exception1-handling-statements

WHEN exception2 THEN

exception2-handling-statements

WHEN exception3 THEN

exception3-handling-statements

WHEN others THEN

exception3-handling-statements

END;

D. Raising Exceptions

- Exceptions are raised by the database server automatically whenever there is any internal database error, but exceptions can be raised explicitly by the programmer by using the command RAISE. Following is the simple syntax for raising an exception:

DECLARE

exception_name EXCEPTION;

BEGIN

IF condition THEN

RAISE exception_name;

END IF;

EXCEPTION

WHEN exception_name THEN statement;

END;

IMPLEMENTATION:

Consider Tables:

1. Borrower (Roll_no, Name, Date of Issue, Name of Book, Status)

2. Fine (Roll_no, Date, Amt)

- Accept Roll_no and 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 5 per 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.
- Also handles the exception by named exception handler or user define exception handler.

OR

Write a **PL/SQL code block** to calculate the area of a circle for a value of radius varying from 5 to 9. Store the radius and the corresponding values of calculated area in an empty table named areas, consisting of two columns, radius and area.

Note: Instructor will frame the problem statement for writing PL/SQL block in line with above statement.

CONCLUSION:

QUESTIONS:

1. What are the advantages of PLSQL over SQL?
2. List Different Pre-defined Exceptions.
3. Explain User-defined exceptions.

MES College of Engineering Pune-01**Department of Computer Engineering**

| | |
|-----------------------------|---------------------------------|
| Name of Student: | Class: |
| Semester/Year: | Roll No: |
| Date of Performance: | Date of Submission: |
| Examined By: | Experiment No: Part A-06 |

PART: A) ASSIGNMENT NO: 06

AIM: Named PL/SQL Block: PL/SQL Stored Procedure and Stored Function.

OBJECTIVES:

- To develop basic, intermediate and advanced Database programming skills.
- To learn the concept of procedural language.
- To learn stored procedure and stored function in PL/SQL.

APPARATUS:

- Operating System recommended: 64-bit Open source Linux or its derivative
- Database: MySQL/ Oracle 11g Database.

THEORY:**A. PL/SQL Stored Procedure**

- 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.
- A procedure may or may not return any value
- 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.
- **Procedures: Passing Parameters**
- ❖ **IN parameters:**
 - ✓ The IN parameter can be referenced by the procedure or function.
 - ✓ This parameter is used for giving input to the subprograms.

- ✓ It is a read-only variable inside the subprograms, their values cannot be changed inside the subprogram
- ❖ **OUT parameters:**
 - ✓ The OUT parameter cannot be referenced by the procedure or function.
 - ✓ This parameter is used for getting output from the subprograms.
 - ✓ It is a read-write variable inside the subprograms, their values can be changed inside the subprograms.
- ❖ **INOUT parameters:**
 - ✓ The INOUT parameter can be referenced by the procedure or function.
 - ✓ This parameter is used for both giving input and for getting output from the subprograms.
 - ✓ It is a read-write variable inside the subprograms, their values can be changed inside the subprograms.
- **PL/SQL Create Procedure**

Syntax for creating procedure:

```
CREATE [OR REPLACE] PROCEDURE procedure_name [ (parameter [,parameter]) ]  
IS  
    [Declaration_section]  
BEGIN  
    Executable_section  
    [EXCEPTION  
    Exception_section]  
END [procedure_name];  
/
```

Syntax for drop procedure

```
DROP PROCEDURE procedure_name
```

B. 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:

```
CREATE [OR REPLACE] FUNCTION function_name [parameters]
[(parameter_name [IN | OUT | IN OUT] type [, ...])]
RETURN return_datatype
{IS | AS}
BEGIN
    < function_body >
END [function_name];
/
```

Syntax for removing your created function:

```
DROP FUNCTION function_name;
```

IMPLEMENTATION:

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 and 900 category is first class, if marks ≥ 899 and ≤ 825 category is Higher Second Class.

Write a PL/SQL block to use procedure created with above requirement.

Stud_Marks(name, total_marks)

Result(Roll, Name, Class)

Note: Instructor will frame the problem statement for writing stored procedure and function in line with above statement.

CONCLUSION:**QUESTIONS:**

1. What is a Stored Procedure?
2. Describe the use of %ROWTYPE and %TYPE in SQL?
3. Explain IN, OUT, IN-OUT mode in stored procedure.
4. What is a Stored Function?
5. What is difference between stored functions and stored procedures?

MES College of Engineering Pune-01
Department of Computer Engineering

| | |
|-----------------------------|---------------------------------|
| Name of Student: | Class: |
| Semester/Year: | Roll No: |
| Date of Performance: | Date of Submission: |
| Examined By: | Experiment No: Part A-07 |

PART: A) ASSIGNMENT NO: 07

AIM: Cursors: (All types: Implicit, Explicit, Cursor FOR Loop, Parameterized Cursor)

OBJECTIVES:

- To develop basic, intermediate and advanced Database programming skills.
- To learn the concept of procedural language.
- To study cursor programming and different cursor operations.

APPRATUS:

- Operating System recommended: 64-bit Open source Linux or its derivative
- Database: MySQL/ Oracle 11g Database.

THEORY:

A. PL/SQL Cursor

- When an SQL statement is processed, Oracle creates a memory area known as context area.
- A cursor is a pointer to this context area.
- It contains all information needed for processing the statement.
- In PL/SQL, the context area is controlled by Cursor.
- A cursor contains information on a select statement and the rows of data accessed by it.
- A cursor can hold more than one row, but can process only one row at a time. The set of rows the cursor holds is called the active set.

PL/SQL Implicit Cursors

- The implicit cursors are automatically generated by Oracle while an SQL statement is executed, if you don't use an explicit cursor for the statement.
- These are created by default to process the statements when DML statements like INSERT, UPDATE, DELETE etc. are executed.

- Oracle provides some attributes known as Implicit cursor's attributes to check the status of DML operations.

%FOUND, %NOTFOUND, %ROWCOUNT and %ISOPEN.

| Attributes | Return Value |
|--|---|
| %FOUND SQL %FOUND | The return value is TRUE , if the DML statements like INSERT, DELETE and UPDATE affect at least one row and if SELECT ...INTO statement return at least one row. |
| | The return value is FALSE , if DML statements like INSERT, DELETE and UPDATE do not affect row and if SELECT...INTO statement do not return a row |
| %NOTFOUND SQL %NOTFOUND | The return value is FALSE , if DML statements like INSERT, DELETE and UPDATE at least one row and if SELECT ...INTO statement return at least one row. |
| | The return value is TRUE , if a DML statement likes INSERT, DELETE and UPDATE do not affect even one row and if SELECT ...INTO statement does not return a row. |
| %ROWCOUNT SQL %ROWCOUNT | Return the number of rows affected by the DML operations INSERT, DELETE, UPDATE, SELECT. |
| %ISOPEN SQL %ISOPEN | It always returns FALSE for implicit cursors, because the SQL cursor is automatically closed after executing its associated SQL statements. |

PL/SQL Explicit Cursors

- The Explicit cursors are defined by the programmers to gain more control over the context area.
- These cursors should be defined in the declaration section of the PL/SQL block. It is created on a SELECT statement which returns more than one row.
- General Syntax for creating a cursor:

CURSOR cursor_name **IS** select_statement;;

- Steps:
 1. Declare the cursor to initialize in the memory.
 2. Open the cursor to allocate memory.
 3. Fetch the cursor to retrieve data.
 4. Close the cursor to release allocated memory.

❖ 1) Declare the cursor:

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

CURSOR name **IS** SELECT statement;

❖ 2) Open the cursor:

It is used to allocate memory for the cursor and make it easy to fetch the rows returned by the SQL statements into it.

OPEN cursor_name;

❖ 3) Fetch the cursor:

It is used to access one row at a time. You can fetch rows from the above-opened cursor as follows:

FETCH cursor_name INTO variable_list;

❖ 4) Close the cursor:

It is used to release the allocated memory. The following syntax is used to close the above-opened cursors.

CLOSE cursor_name;

IMPLEMENTATION:

Write a PL/SQL block of code using parameterized Cursor that will merge the data available in the newly created table N_Roll Call 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.

Note: Instructor will frame the problem statement for writing PL/SQL block using all types of Cursors in line with above statement.

CONCLUSION:**QUESTIONS:**

1. What are different types of cursor? Explain each type with syntax.
2. What are the different attributes of cursor?
3. What is the parameterized cursor?

MES College of Engineering Pune-01
Department of Computer Engineering

| | |
|-----------------------------|---------------------------------|
| Name of Student: | Class: |
| Semester/Year: | Roll No: |
| Date of Performance: | Date of Submission: |
| Examined By: | Experiment No: Part A-08 |

PART: A) ASSIGNMENT NO: 08

AIM: Database Trigger (All Types: Row level and Statement level triggers, Before and After Triggers).

OBJECTIVES:

- To develop basic, intermediate and advanced Database programming skills.
- To learn the concept of procedural language.
- To learn various Database Trigger (All Types: Row level and Statement level triggers, Before and After Triggers).

APPRATUS:

- Operating System recommended: 64-bit Open source Linux or its derivative
- Database: MySQL/ Oracle 11g Database.

THEORY:

PL/SQL Trigger

- A database trigger is a stored procedure that automatically executes whenever an event occurs. The event may be insert-delete-update operations.
- 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 could be defined on the table, view, schema, or database with which the event is associated.
- A procedure is executed explicitly from another block via a procedure call with passing arguments,
- While a trigger is executed (or fired) implicitly whenever the triggering event (DML: INSERT, UPDATE, or DELETE) happens, and a trigger doesn't accept arguments.
- 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).
- **Type of Triggers**
 - ✓ **BEFORE Trigger:** BEFORE trigger execute before the triggering DML statement (INSERT, UPDATE, DELETE) execute. Triggering SQL statement is may or may not execute, depending on the BEFORE trigger conditions block.
 - ✓ **AFTER Trigger:** AFTER trigger execute after the triggering DML statement (INSERT, UPDATE, DELETE) executed. Triggering SQL statement is execute as soon as followed by the code of trigger before performing Database operation.
 - ✓ **ROW Trigger:** ROW trigger fire for each and every record which are performing INSERT, UPDATE, DELETE from the database table. If row deleting is define as trigger event, when trigger file, deletes the five rows each times from the table.
 - ✓ **Statement Trigger:** Statement trigger fire only once for each statement. If row deleting is define as trigger event, when trigger file, deletes the five rows at once from the table.

- **Syntax of 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 | FOR EACH STATEMENT [ WHEN Condition ]
    WHEN (condition)

DECLARE
    Declaration-statements

BEGIN
    Executable-statements

EXCEPTION
    Exception-handling-statements

END;
```

- ✓ **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

IMPLEMENTATION:

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.

Note: Instructor will Frame the problem statement for writing PL/SQL block for all types of Triggers in line with above statement.

CONCLUSION:

QUESTIONS:

1. What is a trigger?
2. What are Benefits of Triggers?
3. What are Row triggers and Statement triggers?
4. Why are we using Before and After triggers?
5. What is Insert, Update and Delete triggers?

MES College of Engineering Pune-01

Department of Computer Engineering

| | |
|-----------------------------|--------------------------------------|
| Name of Student: | Class: |
| Semester/Year: | Roll No: |
| Date of Performance: | Date of Submission: |
| Examined By: | Experiment No: A-09,B-04,C-01 |

ASSIGNMENT - A-09,B-04,C-01

AIM: Database Connectivity & Mini Project

A-09: Write a program to implement MySQL/Oracle database connectivity with any front end language to implement Database navigation operations (add, delete, edit etc.)

B-04: Write a program to implement Mongo DB database connectivity with any front end language to implement Database navigation operations (add, delete, edit etc.)

C-01: Using the database concepts covered in Group A and Group B, develop an application with following details:

1. Follow the same problem statement decided in Assignment-1 of Group A.
2. Follow the Software Development Life cycle and other concepts learnt in Software Engineering Course throughout the implementation.
3. Develop application considering:
 - Front End: Java/Perl/PHP/Python/Ruby/.net/any other language
 - Back End : MongoDB/ MySQL/Oracle
4. Test and validate application using Manual/Automation testing.
5. Student should develop application in group of 2-3 students and submit the Project Report which will consist of documentation related to different phases of Software

Development Life Cycle:

- Title of the Project, Abstract, Introduction
- Software Requirement Specification
- Conceptual Design using ER features, Relational Model in appropriate Normalize form
- Graphical User Interface, Source Code
- Testing document
- Conclusion

OBJECTIVES:

- To develop basic, intermediate and advanced Database programming skills.
- To develop basic Database administration skill.
- To develop the ability to handle databases of varying complexities.
- To use advanced database Programming concepts.

APPARATUS:

- Operating System recommended: 64-bit Open source Linux or its derivative
- Front End: Java/Perl/PHP/Python/Ruby/.net/any other language
- Back End : MongoDB/ MySQL/Oracle

THEORY:

Data Models:

A Data Model is a logical structure of Database. It describes the design of database to reflect entities, attributes, relationship among data, constrains etc.

Data Models Types:

A. Record-based Data Models

- The Relational Model
- The Network Model
- The Hierarchical Model

B. Object-based Data Models

- The E-R Model
- The Object-Oriented Model

C. Physical Data Models

A. Relational Model:

Relational model stores data in the form of tables. This concept purposed by Dr. E.F. Codd, a researcher of IBM in the year 1960s. The relational model consists of three major components:

1. The set of relations and set of domains that defines the way data can be represented (data structure).
2. Integrity rules that define the procedure to protect the data (data integrity).
3. The operations that can be performed on data (data manipulation).

Basic Terminology used in Relational Model

The figure shows a relation with the. Formal names of the basic components marked the entire structure is, as we have said, a relation.

The diagram illustrates a relation structure. At the top, a horizontal line separates the header from the data. Above this line, the word 'Attributes' is written in red, followed by a horizontal line segment, and then the attribute names 'Emp_Code', 'Name', and 'Year' are listed in red. Below the horizontal line, the word 'Tuples' is written in red. To the right of 'Tuples', there are four horizontal arrows pointing to the data rows. Each row contains four values: '21130', 'Amar Jain', '1', and '30745', 'Kuldeep', '3', '41894', 'Manoj', '2', and '51207', 'Rita bajaj', '6'.

| Attributes | Emp_Code | Name | Year |
|------------|----------|------------|------|
| Tuples | 21130 | Amar Jain | 1 |
| | 30745 | Kuldeep | 3 |
| | 41894 | Manoj | 2 |
| | 51207 | Rita bajaj | 6 |

Tuples of a Relation - Each row of data is a tuple.

Cardinality of a relation: The number of tuples in a relation determines its cardinality. In this case, the relation has a cardinality of 4.

Degree of a relation: Each column in the tuple is called an attribute. The number of attributes in a relation determines its degree. The relation in figure has a degree of 3.

Domains: A domain definition specifies the kind of data represented by the attribute.

Keys of a Relation

It is a set of one or more columns whose combined values are *unique* among all occurrences in a given table. A key is the relational means of specifying uniqueness. Some different types of keys are:

Primary key – It is an attribute or a set of attributes of a relation which possess the properties of uniqueness and irreducibility (No subset should be unique).

Foreign key – It is the attributes of a table, which refers to the primary key of some another table. Foreign key permit only those values, which appears in the primary key of the table to which it refers or may be null (Unknown value).

Operations in Relational Model:

The four basic operations Insert, Update, Delete and Retrieve operations.

B. ER Data Model: An Entity Relationship Model

An entity relationship model, also called an entity-relationship (ER) diagram, is a graphical representation of entities and their relationships to each other, typically used in computing in regard to the organization of data within databases or information systems.

E-R Model is based on –

- A. Entities and their attributes.
- B. Relationships among entities.

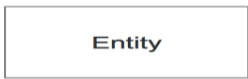



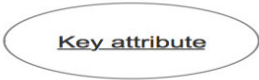


1.Entity – An entity in an ER Model is a “thing” or “object ” in the real-world having properties called attributes. An Entity set is a set of entities of the same type that share the same properties, or attributes.



2. Attributes- Entities are represented by means of their properties, called attributes. All attributes have values. For example, a student entity may have name, class, and age as attributes.

ER diagram: An ER diagram is a pictorial representation of the information that can be captured by a database. Such a picture serves two purposes:





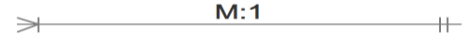
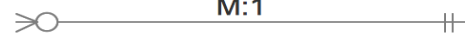
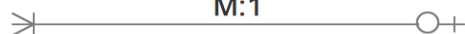
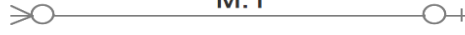
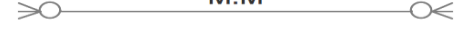




1. It allows database professionals to describe an overall design concisely yet accurately.
2. It can be easily transformed into the relational schema.

Entity Relationship Diagram Symbols — Chen notation

| Symbol | Shape Name | Symbol Description |
|---|-----------------------|--|
| Entities | | |
|  | Entity | An entity is represented by a rectangle which contains the entity's name. |
|  | Weak Entity | An entity that cannot be uniquely identified by its attributes alone. The existence of a weak entity is dependent upon another entity called the owner entity. The weak entity's identifier is a combination of the identifier of the owner entity and the partial key of the weak entity. |
|  | Associative Entity | An entity used in a many-to-many relationship (represents an extra table). All relationships for the associative entity should be many |
| Attributes | | |
|  | Attribute | In the Chen notation, each attribute is represented by an oval containing attributes name |
|  | Key attribute | An attribute that uniquely identifies a particular entity. The name of a key attribute is underscored. |
|  | Multivalued attribute | An attribute that can have many values (there are many distinct values entered for it in the same column of the table). Multivalued attribute is depicted by a dual oval. |
|  | Derived attribute | An attribute whose value is calculated (derived) from other attributes. The derived attribute may or may not be physically stored in the database. In |

| | | |
|---|---------------------------------|--|
| | | the Chen notation, this attribute is represented by dashed oval. |
| Relationships | | |
|  | Strong relationship | A relationship where entity is existence-independent of other entities and PK of Child doesn't contain PK component of Parent Entity. A strong relationship is represented by a single rhombus |
|  | Weak (identifying) relationship | A relationship where Child entity is existence-dependent on parent, and PK of Child Entity contains PK component of Parent Entity. This relationship is represented by a double rhombus. |

Entity Relationship Diagram Symbols — Crow's Foot notation

| Symbol | Meaning |
|---|--|
|  | Zero or One |
|  | One or More |
|  | One and only One |
|  | Zero or More |
|  | a one through many notation on one side of a relationship and a one and only one on the other |
|  | a zero through many notation on one side of a relationship and a one and only one on the other |
|  | a one through many notation on one side of a relationship and a zero or one notation on the other |
|  | a zero through many notation on one side of a relationship and a zero or one notation on the other |
|  | a zero through many on both sides of a relationship |
|  | a zero through many on one side and a one through many on the other |
|  | a one through many on both sides of a relationship |
|  | a one and only one notation on one side of a relationship and a zero or one on the other |
|  | a one and only one notation on both sides |

Transform ER Diagram into Tables:

Since ER diagram gives us the good knowledge about the requirement and the mapping of the entities in it, we can easily convert them as tables and columns. i.e.; using ER diagrams one can easily create relational data model, which nothing but the logical view of the database.

There are various steps involved in converting it into tables and columns. Each type of entity, attribute and relationship in the diagram takes their own depiction here. Consider the ER diagram below and will see how it is converted into tables, columns and mappings.

The basic rule for converting the ER diagrams into tables is

- Convert all the Entities in the diagram to tables.

All the entities represented in the rectangular box in the ER diagram become independent tables in the database. In the below diagram, STUDENT, COURSE, LECTURER and SUBJECTS forms individual tables.

- All single valued attributes of an entity is converted to a column of the table

All the attributes, whose value at any instance of time is unique, are considered as columns of that table. In the STUDENT Entity, STUDENT_ID, STUDENT_NAME form the columns of STUDENT table. Similarly, LECTURER_ID, LECTURER_NAME form the columns of LECTURER table. And so on.

- Key attribute in the ER diagram becomes the Primary key of the table.

In diagram above, STUDENT_ID, LECTURER_ID, COURSE_ID and SUB_ID are the key attributes of the entities. Hence we consider them as the primary keys of respective table.

- Declare the foreign key column, if applicable.

In the diagram, attribute COURSE_ID in the STUDENT entity is from COURSE entity. Hence add COURSE_ID in the STUDENT table and assign it foreign key constraint. COURSE_ID and SUBJECT_ID in LECTURER table forms the foreign key column. Hence by declaring the foreign key constraints, mapping between the tables are established.

- Any multi-valued attributes are converted into new table.

A hobby in the Student table is a multivalued attribute. Any student can have any number of hobbies. So we cannot represent multiple values in a single column of STUDENT table. We need to store it separately, so that we can store any number of hobbies, adding/ removing / deleting hobbies should not create any redundancy or anomalies in the system. Hence we create a separate table STUD_HOBBY with STUDENT_ID and HOBBY as its columns.

We create a composite key using both the columns

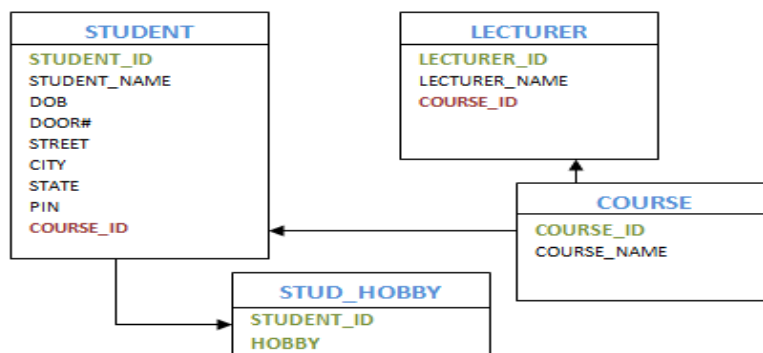
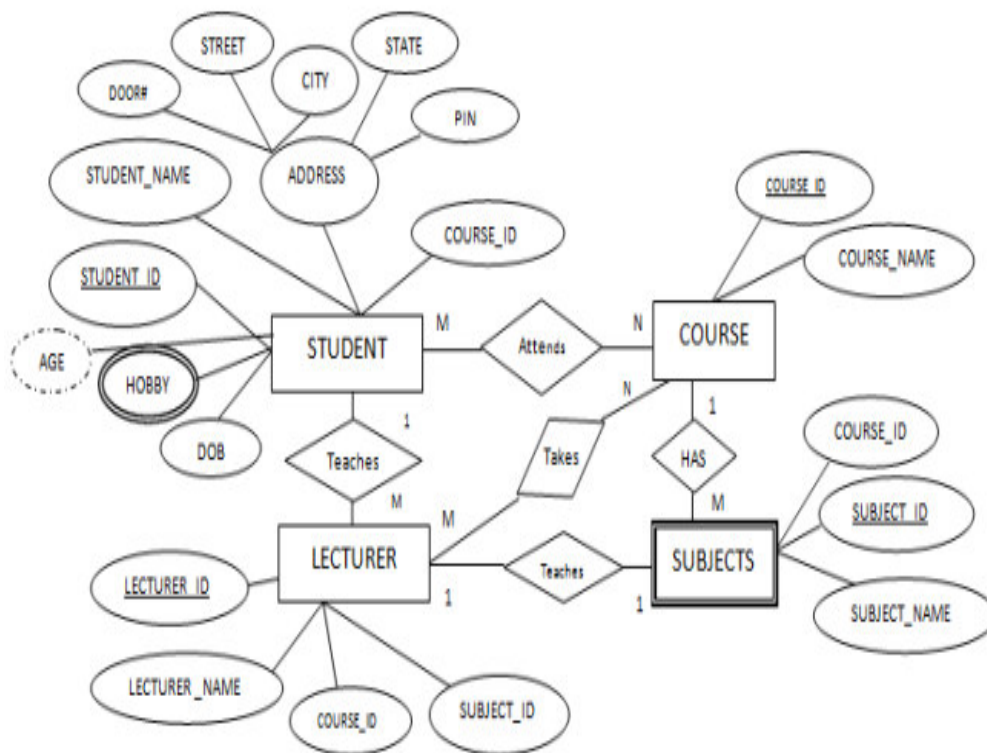
Any composite attributes are merged into same table as different columns.

In the diagram above, Student Address is a composite attribute. It has Door#, Street, City, State and Pin. These attributes are merged into STUDENT table as individual columns.

- One can ignore derived attribute, since it can be calculated at any time.

In the STUDENT table, Age can be derived at any point of time by calculating the difference between DateOfBirth and current date. Hence we need not create a column for this attribute. It reduces the duplicity in the database.

These are the very basic rules of converting ER diagram into tables and columns, and assigning the mapping between the tables. Table structure at this would be as below:



CONCLUSION:

MES College of Engineering Pune-01**Department of Computer Engineering**

| | |
|-----------------------------|---------------------------------|
| Name of Student: | Class: |
| Semester/Year: | Roll No: |
| Date of Performance: | Date of Submission: |
| Examined By: | Experiment No: Part B-01 |

GROUP: B) ASSIGNMENT NO: 01**AIM: MongoDB Queries:**

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

OBJECTIVES:

- To develop basic, intermediate and advanced Database programming skills.
- To develop basic Database administration skill.
- To Study NoSQL database.
- To Study document oriented database-Mongodb.

APPARATUS:

- Operating System recommended: 64-bit Open source Linux or its derivative.
- Front End: Java/PHP/Python
- Back End: MongoDB

THEORY:**1. What is MongoDB?**

- MongoDB is an open-source document database that provides high performance, high availability, and automatic scaling.
- A record in MongoDB is a document, which is a data structure composed of field and value pairs.
- MongoDB documents are BSON documents. BSON is a binary representation of JSON with additional type information.
- A document is the basic unit of data for MongoDB and is roughly equivalent to a row in a relational database management system.

- A collection is a group of documents. If a document is the MongoDB analog of a row in a relational database, then a collection are analog to a table.
- MongoDB is type-sensitive and case-sensitive.
- Every document has a special key, "_id", that is unique within a collection.
- In the documents, the value of a field can be any of the BSON data types, including other documents, arrays, and arrays of documents.

```
{  
  Name: "Sagar",  
  Class: "TE Comp",  
  College: "MESCOE",  
  Age: 26,  
  Subject: ["DMSA", "OSD", "FCA", "DCWSN", "TOC"]  
}
```

Fig: MongoDB document

2. MongoDB 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

Example: >use mydb

switched to db mydb

- ✓ To check your currently selected database uses the command db.

>db

mydb

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

>show dbs

local 0.78125GB

test 0.23012GB

- ✓ Your created database (mydb) is not present in list. To display database you need to insert atleast one document into it.
- ✓ In mongodb default database is test. If you didn't create any database then collections will be stored in test database.

3. MongoDB Drop Database

- **The dropDatabase () Method**

MongoDB db.dropDatabase () command is used to drop a existing database.

Syntax: db.dropDatabase()

This will delete the selected database. If you have not selected any database, then it will delete default 'test' database.

4. MongoDB Create Collection

- **The createCollection() Method**

MongoDB db.createCollection(name, options) is used to create collection. In the command, name is name of collection to be created. Options are a document and used to specify configuration of collection. Options parameter is optional, so you need to specify only name of the collection.

Syntax: db.createCollection(name, options)

Examples: Basic syntax of createCollection() method without options is as follows:

```
>use test  
switched to db test  
>db.createCollection("mycollection")  
{ "ok" : 1 }
```

5. MongoDB Drop Collection

- **The drop() Method**

MongoDB's db.collection.drop() is used to drop a collection from the database.

Syntax: db.COLLECTION_NAME.drop()

6. MongoDB - Insert Document

- **The insert() Method**

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

Syntax: db.COLLECTION_NAME.insert(document)

7. MongoDB - Query Document

- **The find() Method**

To query data from MongoDB collection, you need to use MongoDB's find() method. Find() method will display all the documents in a non-structured way.

Syntax: db.COLLECTION_NAME.find()

- **The pretty() Method**

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

Syntax: `db.COLLECTION_NAME.find().pretty()`

- **The forEach() Method**

To display the results in a JSON format, you can use `forEach()` method.

Syntax: `db.COLLECTION_NAME.find().forEach(printjson)`

8. MongoDB Update Document

MongoDB's `update()` and `save()` methods are used to update document into a collection.

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

Syntax:

`db.COLLECTION_NAME.update(SELECTIOIN_CRITERIA,UPDATED_DATA)`

- **MongoDB Save() Method**

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

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

9. MongoDB Delete Document

- **The remove() Method**

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

1. Deletion criteria: (Optional) deletion criteria according to documents will be removed.

2. `justOne`: (Optional) if set to true or 1, then remove only one document.

Syntax: `db.COLLECTION_NAME.remove(DELLETION_CRITTERIA)`

IMPLEMENTATION:

A. Create Empdb database

B. Create Employee collection by considering following Fields:

- Empid: Number
- Name: Embedded Doc (FName, LName)
- Company Name: String
- Salary: Number
- Designation: String

- vi. Age: Number
 - vii. Expertise: Array
 - viii. DOB: String or Date
 - ix. Email id: String
 - x. Contact: String
 - xi. Address: Array of Embedded Doc (PAddr, LAddr)
- C. Insert at least 10 documents in Employee Collection and execute following statements:
- 1. Select all documents where the Designation field has the value "Programmer" and the value of the salary field is greater than 30000.
 - 2. Creates a new document if no document in the employee collection contains
 - 3. {Designation: "Tester", Company_name: "TCS", Age: 25}
 - 4. Selects all documents in the collection where the field age has a value less than 30 or the value of the salary field is greater than 40000.
 - 5. Matches all documents where the value of the field Address is an embedded document that contains only the field city with the value "Pune" and the field Pin_code with the value "411001".
 - 6. Finds all documents with Company_name: "TCS" and modifies their salary field by 2000.
 - 7. Find documents where Designation is not equal to "Developer".
 - 8. Find _id, Designation, Address and Name from all documents where Company_name is "Infosys".
 - 9. Selects all documents in the employee collection where the value of the Designation is either "Developer" or "Tester".
 - 10. Find all document with Exact Match on an Array having Expertise: ['Mongodb','Mysql', 'Cassandra']
 - 11. Drop Single documents where designation="Developer"

CONCLUSION:

QUESTIONS:

- 1. What is NoSQL and enlist its benefits.
- 2. Shows the relationship of RDBMS terminology with MongoDB.

3. Explain CRUD operations in MongoDB database with suitable Example
4. What are Advantages of MongoDB over RDBMS?
5. Enlist Basic datatypes of MongoDB.
6. What is different between SAVE and UPDATE method.
7. What is ObjectId in MongoDB?
8. Explain different method to insert document in MongoDB.
9. Explain CAP & BASE Theorem in NoSQL with Suitable Example.
10. What are different key features of MongoDB.

MES College of Engineering Pune-01**Department of Computer Engineering**

| | |
|-----------------------------|---------------------------------|
| Name of Student: | Class: |
| Semester/Year: | Roll No: |
| Date of Performance: | Date of Submission: |
| Examined By: | Experiment No: Part B-02 |

GROUP: B) ASSIGNMENT NO: 02**AIM: MongoDB – Aggregation and Indexing:**

Design and Develop MongoDB Queries using aggregation and indexing with suitable example using MongoDB.

OBJECTIVES:

- To develop basic, intermediate and advanced Database programming skills.
- To develop basic Database administration skill.
- To learn aggregation and indexing for NoSQL database.

APPARATUS:

- Operating System recommended: 64-bit Open source Linux or its derivative.
- Front End: Java/PHP/Python.
- Back End: MongoDB.

THEORY:**A. MongoDB 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.
- Running data aggregation on the mongod instance simplifies application code and limits resource requirements.
- Like queries, aggregation operations in MongoDB use collections of documents as an input and return results in the form of one or more documents.

- MongoDB provides three ways to perform aggregation: the aggregation pipeline, the map-reduce function and single purpose aggregation methods and commands.

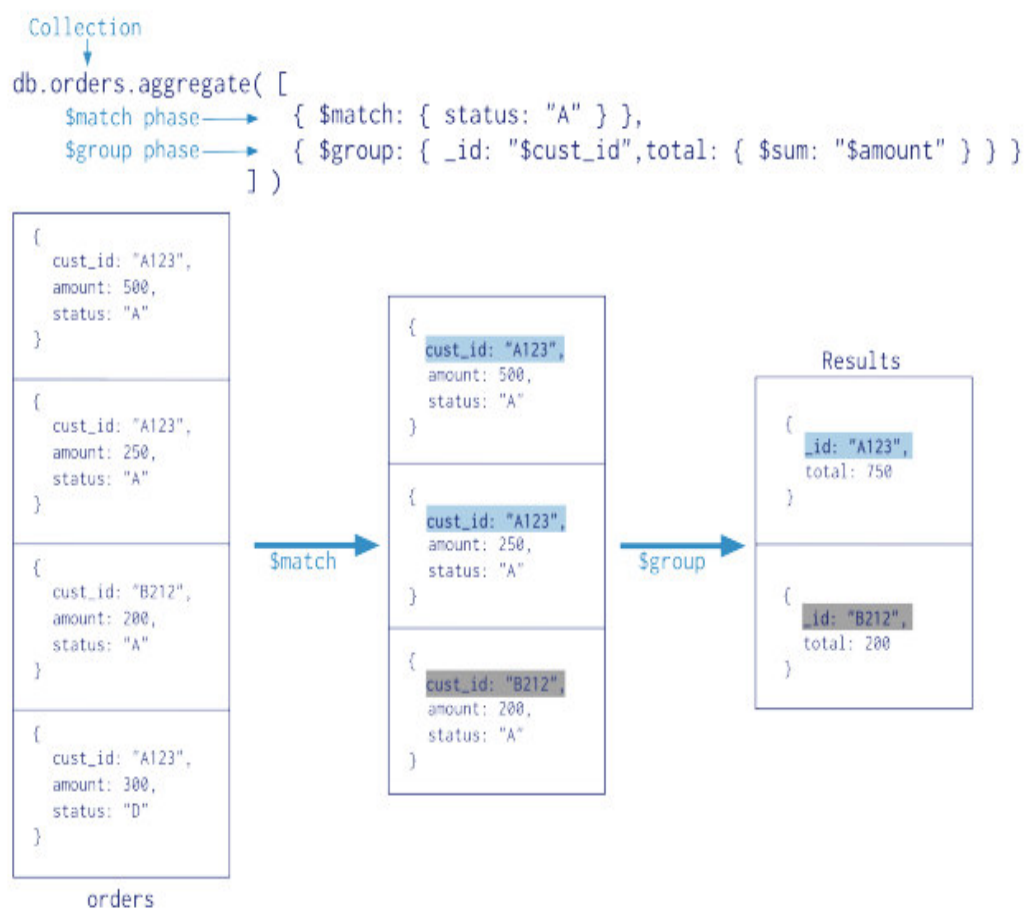
1. The aggregate() Method

For the aggregation in mongodb you should use **aggregate()** method.

Syntax: `db.COLLECTION_NAME.aggregate(AGGREGATE_OPERATION)`

2. Aggregation Pipeline

- The aggregation pipeline is a framework for data aggregation modeled on the concept of data processing pipelines.
- The aggregation pipeline provides an alternative to map-reduce and may be the preferred solution for many aggregation tasks where the complexity of map-reduce may be unwarranted.
- In UNIX command shell pipeline means the possibility to execute an operation on some input and use the output as the input for the next command and so on.
- There is a set of possible stages and each of those is taken a set of documents as an input and is producing a resulting set of documents (or the final resulting JSON document at the end of the pipeline). This can then in turn again be used for the next stage and so on.



3. Pipeline Operators

- **\$project:** Reshapes a document stream. \$project can rename, add, or remove fields as well as create computed values and sub-documents.
- **\$match:** Filters the document stream, and only allows matching documents to pass into the next pipeline stage. \$match uses standard MongoDB queries.
- **\$redact:** Restricts the content of a returned document on a per-field level.
- **\$limit:** Restricts the number of documents in an aggregation pipeline.
- **\$skip:** Skips over a specified number of documents from the pipeline and returns the rest.
- **\$unwind:** Takes an array of documents and returns them as a stream of documents.
- **\$group:** Groups documents together for the purpose of calculating aggregate values based on a collection of documents.
- **\$sort:** Takes all input documents and returns them in a stream of sorted documents.
- **\$geoNear:** Returns an ordered stream of documents based on proximity to a geospatial point.
- **\$out:** Writes documents from the pipeline to a collection. The \$out operator must be the last stage in the pipeline.

4. Expression Operators

a) \$group Operators

- **\$addToSet:** Returns an array of all the unique values for the selected field among for each document in that group.
- **\$first:** Returns the first value in a group.
- **\$last:** Returns the last value in a group.
- **\$max:** Returns the highest value in a group.
- **\$min:** Returns the lowest value in a group.
- **\$avg:** Returns an average of all the values in a group.
- **\$push:** Returns an array of all values for the selected field among for each document in that group.
- **\$sum:** Returns the sum of all the values in a group.

b) Comparison Operators

- **\$cmp:** Compares two values and returns the result of the comparison as an integer.
- **\$eq:** Takes two values and returns true if the values are equivalent.
- **\$gt:** Takes two values and returns true if the first is larger than the second.

- **\$gte**: Takes two values and returns true if the first is larger than or equal to the second.
- **\$lt**: Takes two values and returns true if the second value is larger than the first.
- **\$lte**: Takes two values and returns true if the second value is larger than or equal to the first.
- **\$ne**: Takes two values and returns true if the values are not equivalent.

c) Boolean Operators

- **\$and**: Returns true only when all values in its input array are true.
- **\$or**: Returns true when any value in its input array are true.
- **\$not**: Returns the Boolean value that is the opposite of the input value.

d) Arithmetic Operators

- **\$add**: Computes the sum of an array of numbers.
- **\$divide**: Takes two numbers and divides the first number by the second.
- **\$mod**: Takes two numbers and calculates the modulo of the first number divided by the second.
- **\$multiply**: computes the product of an array of numbers.
- **\$subtract**: Takes an array that contains two numbers or two dates and subtracts the second value from the first.

e) Array Operators

- **\$size**: Returns the size of the array.

f) Date Operators

- **\$dayOfYear**: Converts a date to a number between 1 and 366.
- **\$dayOfMonth**: Converts a date to a number between 1 and 31.
- **\$dayOfWeek**: Converts a date to a number between 1 and 7.
- **\$year**: Converts a date to the full year.
- **\$month**: Converts a date into a number between 1 and 12.
- **\$week**: Converts a date into a number between 0 and 53
- **\$hour**: Converts a date into a number between 0 and 23.
- **\$minute**: Converts a date into a number between 0 and 59.
- **\$second**: Converts a date into a number between 0 and 59. May be 60 to account for leap seconds.
- **\$millisecond**: Returns the millisecond portion of a date as an integer between 0 and 999.

B. MongoDB Indexing

- Indexes support the efficient resolution of queries.
- Indexes provide high performance read operations for frequently used 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 requires the mongod 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 index.
- Indexes support the efficient execution of queries in MongoDB.
- Without indexes, MongoDB must scan every document in a collection to select those documents that match the query statement.
- These collection scans are inefficient because they require mongod to process a larger volume of data than an index for each operation.

1. The ensureIndex() Method

- To create an index you need to use ensureIndex() method of mongod.
- **Syntax:** `db.COLLECTION_NAME.ensureIndex({KEY:1})`
- Here key is the name of 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.
- **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})
```
- The ensureIndex() method also accepts list of options (which are optional), whose list is given below:
 - ✓ **Background:** (type: Boolean) Builds the index in the background so that building an index does not block other database activities. Specify true to build in the background. The default value is **false**.
 - ✓ **Unique:** (type: Boolean) Creates a unique index so that the collection will not accept insertion of documents where the index key or keys match an existing value in the index. Specify true to create a unique index. The default value is **false**.
 - ✓ **Name:** (type: string) the name of the index. If unspecified, MongoDB generates an index name by concatenating the names of the indexed fields and the sort order.

- ✓ **dropDups** (type: Boolean) Creates a unique index on a field that may have duplicates. MongoDB indexes only the first occurrence of a key and removes all documents from the collection that contain subsequent occurrences of that key. Specify true to create unique index. The default value is **false**.
- ✓ **Sparse:** (type: Boolean) If true, the index only references documents with the specified field. These indexes use less space but behave differently in some situations (particularly sorts). The default value is **false**.
- ✓ **expireAfterSeconds:** (type: integer) Specifies a value, in seconds, as a TTL to control how long MongoDB retains documents in this collection.
- ✓ **v:** (type: index version) The index version number. The default index version depends on the version of mongod running when creating the index.
- ✓ **Weights:** (type: document) The weight is a number ranging from 1 to 99,999 and denotes the significance of the field relative to the other indexed fields in terms of the score.
- ✓ **default_language:** (type: string) for a text index, the language that determines the list of stop words and the rules for the stemmer and tokenizer. The default value is **english**.
- ✓ **language_override:** (type: string) for a text index, specify the name of the field in the document that contains, the language to override the default language. The default value is language.

2. Unique Indexes

- A unique index causes MongoDB to reject all documents that contain a duplicate value for the indexed field.
- To create a unique index, use the `db.collection.ensureIndex()` method with the unique option set to true.
- By default, unique is false on MongoDB indexes.

```
db.members.ensureIndex( { "user_id": 1 }, { unique: true } )
```

3. Drop Duplicates

Force MongoDB to create a unique index by deleting documents with duplicate values when building the index.

```
db.collection.ensureIndex( { a: 1 }, { unique: true, dropDups: true } )
```

4. Remove Index

- To remove an index from a collection use the `dropIndex()` method and the following procedure.

- Remove a Specific Index

```
db.accounts.dropIndex( { "user_id": 1 } )
```

- Remove All Indexes except for the _id index from a collection

```
db.collection.dropIndexes()
```

5. Return a List of All Indexes

- List all Indexes on a Collection
- To return a list of all indexes on a collection, use the db.collection.getIndexes() method.
- Example: To view all indexes on the user collection:

```
db.user.getIndexes()
```

- List all Indexes for a Database
- To return a list of all indexes on all collections in a database:

```
db.system.indexes.find()
```

IMPLEMENTATION:

A. Use Employee database created in Assignment B-01 and perform following aggregation operation

1. Return Designation with Total Salary is Above 200000
2. Find Employee with Total Salary for Each City with Designation="DBA"
3. Find Total Salary of Employee with Designation="DBA" for Each Company
4. Returns names and _id in upper case and in alphabetical order.
5. Count all records from collection
6. For each unique Designation, find avg Salary and output is sorted by AvgSal
7. Return separates value in the Expertise array where Name of Employee="Swapnil"
8. Return separates value in the Expertise array and return sum of each element of array
9. Return Array for Designation whose address is "Pune"
10. Return Max and Min Salary for each company.

B. Use Employee database created in Assignment B-01 and perform following indexing operation

1. To Create Single Field Indexes on Designation
2. To Create Compound Indexes on Name: 1, Age: -1
3. To Create Multikey Indexes on Expertise array
4. Return a List of All Indexes on Collection

5. Rebuild Indexes
6. Drop Index on Remove Specific Index
7. Remove All Indexes except for the _id index from a collection

CONCLUSION:

QUESTIONS:

1. Which are different aggregation commands and aggregation methods?
2. Enlist user-defined and system variables in aggregation.
3. Describe SQL to aggregation Mapping Chart.
4. Explain Indexing Methods in the mongo Shell.
5. What is different option for indexing?
6. Enlist different Pipeline Operators, Expression Operators, and Comparison Operators.
7. What is use of Drop Duplicates option in Indexing?
8. Write method to return a list of all indexes on a collection and databases.

MES College of Engineering Pune-01**Department of Computer Engineering**

| | |
|-----------------------------|---------------------------------|
| Name of Student: | Class: |
| Semester/Year: | Roll No: |
| Date of Performance: | Date of Submission: |
| Examined By: | Experiment No: Part B-03 |

GROUP: B) ASSIGNMENT NO: 03**AIM: MongoDB – Map-reduces operations:**

Implement Map reduces operation with suitable example using MongoDB.

OBJECTIVES:

- To develop basic, intermediate and advanced Database programming skills.
- To develop basic Database administration skill.
- To understand concepts of map reduce in aggregation of NoSQL database MongoDB.

APPRATUS:

- Operating System recommended: 64-bit Open source Linux or its derivative.
- Front End: Java/PHP/Python.
- Back End: MongoDB.

THEORY:**1. MongoDB Aggregation**

- Aggregations are operations that process data records and return computed results.
- MongoDB provides a rich set of aggregation operations that examine and perform calculations on the data sets.
- Running data aggregation on the mongod instance simplifies application code and limits resource requirements.
- Like queries, aggregation operations in MongoDB use collections of documents as an input and return results in the form of one or more documents.

2. Map-Reduce

- Map-reduce is a data processing paradigm for condensing large volumes of data into useful aggregated results.
- For map-reduce operations, MongoDB provides the mapReduce database command.

Syntax: db.COLLECTION_NAME.mapReduce()

- In general, map-reduce operations have two phases: a map stage that processes each document and emits one or more objects for each input document, and reduce phase that combines the output of the map operation.
- Optionally, map-reduce can have a finalize stage to make final modifications to the result. Like other aggregation operations, map-reduce can specify a query condition to select the input documents as well as sort and limit the results.
- All map-reduce functions in MongoDB are JavaScript and run within the mongod process. Map-reduce operations take the documents of a single collection as the input and can perform any arbitrary sorting and limiting before beginning the map stage.
- MapReduce can return the results of a map-reduce operation as a document, or may write the results to collections. The input and the output collections may be sharded.

3. MongoDB and MapReduce

In MapReduce we use mapreduce, map, and reduce keys. These three keys are required, but there are many optional keys that can be passed to the MapReduce command:

- *"finalize" : function*
 - ✓ A final step to send reduce's output to.
- *"keeptemp" : boolean*
 - ✓ If the temporary result collection should be saved when the connection is closed.
- *"out" : string*
 - ✓ Name for the output collection. Setting this option implies keep temp : true.
- *"query" : document*
 - ✓ Query to filter documents by before sending to the map function.
- *"sort" : document*
 - ✓ Sort to use on documents before sending to the map (useful in conjunction with the limit option).
- *"limit" : integer*
 - ✓ Maximum number of documents to send to the map function.
- *"scope" : document*
 - ✓ Variables that can be used in any of the JavaScript code.
- *"verbose" : boolean*
 - ✓ Whether or not to use more verbose output in the server logs.

4. Aggregation Commands

- **Aggregate:** Performs aggregation tasks such as group using the aggregation framework.

- **Count:** Counts the number of documents in a collection.
- **Distinct:** Displays the distinct values found for a specified key in a collection.
- **Group:** Groups documents in a collection by the specified key and performs simple aggregation.
- **mapReduce:** Performs map-reduce aggregation for large data sets.

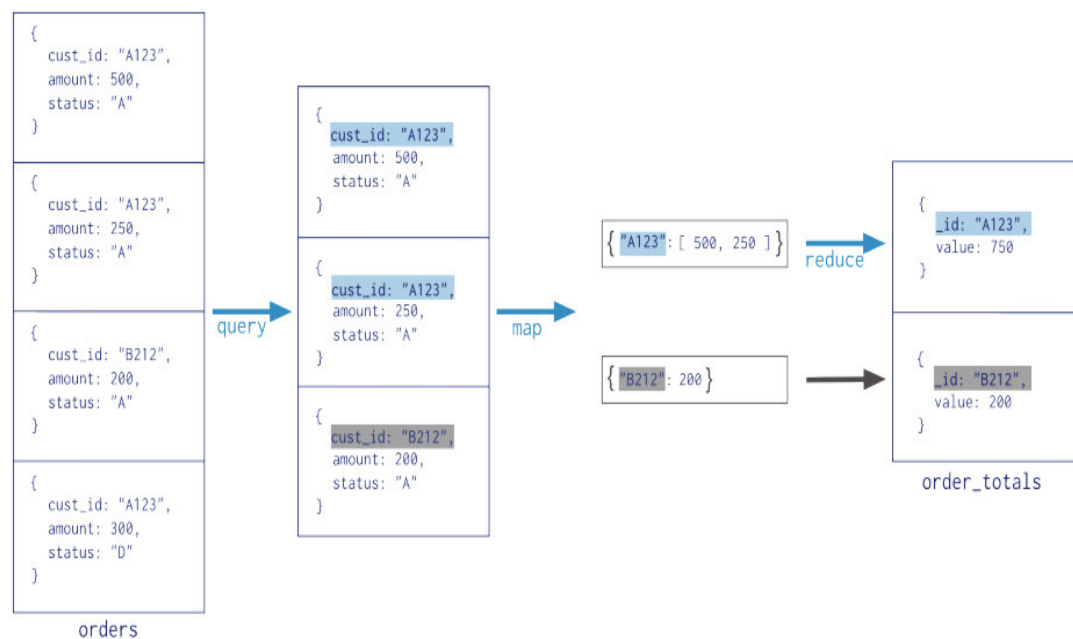
5. Aggregation Methods

- **db.collection.aggregate():** Provides access to the aggregation pipeline.
- **db.collection.group():** Groups documents in a collection by the specified key and performs simple aggregation.
- **db.collection.mapReduce():** Performs map-reduce aggregation for large data sets.

```

Collection
  ↓
db.orders.mapReduce(
  map   → function() { emit( this.cust_id, this.amount ); },
  reduce → function(key, values) { return Array.sum( values ); },
  {
    query → { status: "A" },
    output → "order_totals"
  }
)

```



6. Example for MapReduce Operation

In the mongo shell, the **db.collection.mapReduce()** method is a wrapper around the `mapReduce` command.

The following examples use the **db.collection.mapReduce()** method:

Consider the following map-reduce operations on a collection `orders` that contains documents of the following prototype:

```
{
  _id: ObjectId("50a8240b927d5d8b5891743c"),
  cust_id: "abc123",
  ord_date: new Date("Oct 04, 2012"),
  status: 'A',
  price: 25,
  items: [ { sku: "mmm", qty: 5, price: 2.5 },
    { sku: "nnn", qty: 5, price: 2.5 } ]
}
```

Return the Total Price Per Customer

Perform the map-reduce operation on the orders collection to group by the cust_id, and calculate the sum of the price for each cust_id:

a) Define the map function to process each input document:

- In the function, this refers to the document that the map-reduce operation is processing.
- The function maps the price to the cust_id for each document and emits the cust_id and price pair.

```
var mapFunction1 = function() {
    emit(this.cust_id, this.price);
};
```

b) Define the corresponding reduce function with two arguments keyCustId and valuesPrices:

- The valuesPrices is an array whose elements are the price values emitted by the map function and grouped by keyCustId.
- The function reduces the valuesPrice array to the sum of its elements.

```
var reduceFunction1 = function(keyCustId, valuesPrices) {
    return Array.sum(valuesPrices);
};
```

c) Perform the map-reduce on all documents in the orders collection using the mapFunction1 map function and the reduceFunction1 reduce function.

```
db.orders.mapReduce(
    mapFunction1,
    reduceFunction1,
    { out: "map_reduce_example" }
```

)

This operation outputs the results to a collection named `map_reduce_example`. If the `map_reduce_example` collection already exists, the operation will replace the contents with the results of this map-reduce operation.

IMPLEMENTATION:

Use Employee database created in Assignment B-01 and perform Map reduces operation for following statements:

1. Return the Total Salary of per Company
2. Return the Total Salary of Company Name:"TCS"
3. Return the Avg Salary of Company whose address is "Pune".
4. Return the Total Salary for each Designation of Infosys.
5. Return total count for "State=AP"
6. Return Count for State AP and Age greater than 40.

CONCLUSION:**QUESTIONS:**

1. What are different Aggregation commands?
2. What is map and reduce phase?
3. Explain Map Reduce Concurrency.
4. Write Step for MapReduce Operation with example.
5. What is Map-Reduce JavaScript Function?