

**JAYAWANTRAO SAWANT COLLEGE OF  
ENGINEERING, HADAPSAR, PUNE-28.**

**DEPARTMENT OF COMPUTER ENGINEERING**

***COURSE: TE [2015]***

***DATABASE MANAGEMENT  
SYSTEM LAB MANUAL***



***PREPARED BY:***

***PROF. NIKHILKUMAR B S [M.TECH CSE, PHD(REG.)]***

***PROF. V.B.KHEDEKAR [M.TECH CSE, PHD(REG.)]***

***PROF. S. R. GHULE [ME COMPUTER]***

*Version 2.0*

## Table of Contents

<b>Sr. No.</b>	<b>Topic</b>	<b>Page. No.</b>
1.	<b>Vision, Mission, Quality Policy</b>	4
2	<b>Syllabus</b>	5
2.	<b>How to Use This Manual</b>	7
3.	<b>PEOs and Pos</b>	9
4.	<b>Course Objective</b>	11
5.	<b>Laboratory Objective</b>	
6.	<b>Experiment Learning Outcome (ELO)</b>	13
7.	<b>Lab Plan</b>	
8.	References	

---

## Vision

“To satisfy the aspirations of youth force, who wants to lead Nation towards prosperity through techno-economic development.”

## Mission

“To provide, nurture and maintain an environment of high academic excellence, research and entrepreneurship for all aspiring students, which will prepare them to face global challenges maintaining high ethical and moral standards.”

---

## Syllabus DBMSL

<b>Group A- Database Programming Languages – SQL, PL/SQL</b>	
<b>1.</b>	Study of Open Source Relational Databases : MySQL
<b>2.</b>	Design and Develop SQL DDL statements which demonstrate the use of SQL objects such as Table, View, Index, Sequence, Synonym
<b>3.</b>	Design at least 10 SQL queries for suitable database application using SQL DML statements: Insert, Select, Update, Delete with operators, functions, and set operator.
<b>4.</b>	Design at least 10 SQL queries for suitable database application using SQL DML statements: all types of Join, Sub-Query and View.
<b>5.</b>	<p>Unnamed PL/SQL code block: Use of Control structure and Exception handling is mandatory. Write a PL/SQL block of code for the following requirements:-</p> <p>Schema:</p> <ol style="list-style-type: none"><li>1. Borrower(Roll_no, Name, DateofIssue, NameofBook, Status)</li><li>2. Fine(Roll_no,Date,Amt)</li></ol> <ul style="list-style-type: none"><li><input type="checkbox"/> Accept roll_no &amp; name of book from user.</li><li><input type="checkbox"/> Check the number of days (from date of issue), if days are between 15 to 30 then fine amount will be Rs 5per day.</li><li><input type="checkbox"/> If no. of days&gt;30, per day fine will be Rs 50 per day &amp; for days less than 30, Rs. 5 per day.</li><li><input type="checkbox"/> After submitting the book, status will change from I to R.</li><li><input type="checkbox"/> If condition of fine is true, then details will be stored into fine table.</li></ul> <p><b>Frame the problem statement for writing PL/SQL block inline with above statement.</b></p>
<b>6.</b>	<p>Cursors: (All types: Implicit, Explicit, Cursor FOR Loop, Parameterized Cursor)</p> <p>Write a PL/SQL block of code using parameterized Cursor, that will merge the data available in the newly created table N_RollCall with the data available in the table O_RollCall. If the data in the first table already exist in the second table then that data should be skipped.</p> <p><b>Frame the separate problem statement for writing PL/SQL block to implement all types of Cursors inline with above statement. The problem statement should clearly state the requirements.</b></p>

7.	<p>PL/SQL Stored Procedure and Stored Function.</p> <p>Write a Stored Procedure namely proc_Grade for the categorization of student. if marks scored by students in examination is <math>\leq 1500</math> and marks <math>\geq 990</math> 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</p> <p>Write a PL/SQL block for using procedure created with above requirement.</p> <p>Stud_Marks(name, total_marks) Result(Roll, Name, Class)</p> <p><b>Frame the separate problem statement for writing PL/SQL Stored Procedure and function, inline with above statement. The problem statement should clearly state the requirements.</b></p>
8.	<p>Database Trigger (All Types: Row level and Statement level triggers, Before and After Triggers). Write a database trigger on Library table. The System should keep track of the records that are being updated or deleted. The old value of updated or deleted records should be added in Library_Audit table.</p> <p><b>Frame the problem statement for writing Database Triggers of all types, in-line with above statement. The problem statement should clearly state the requirements.</b></p>
<b>Group B Large Scale Databases</b>	
1.	Study of Open Source NOSQL Database: MongoDB (Installation, Basic CRUD operations, Execution)
2.	Design and Develop MongoDB Queries using CRUD operations. (Use CRUD operations, SAVE method, logical operators)
3.	Implement aggregation and indexing with suitable example using MongoDB.
4.	Implement Map reduces operation with suitable example using MongoDB.
5.	Design and Implement any 5 query using MongoDB
6.	Create simple objects and array objects using JSON
7.	Encode and Decode JSON Objects using Java/Perl/PHP/Python/Ruby
<b>Group C Mini Project : Database Project Life Cycle</b>	
1.	Write a program to implement MogoDB database connectivity with PHP/ python/Java Implement Database navigation operations (add, delete, edit etc. ) using ODBC/JDBC.
2.	Implement MYSQL/Oracle database connectivity with PHP/ python/Java Implement Database navigation operations (add, delete, edit,) using ODBC/JDBC.
3.	<p>Using the database concepts covered in Part-I &amp; Part-II &amp; connectivity concepts covered in Part C, students in group are expected to design and develop database application with following details:</p> <p><b>Requirement Gathering and Scope finalization</b></p> <p><b>Database Analysis and Design:</b></p> <p><input type="checkbox"/> Design Entity Relationship Model, Relational Model, Database Normalization</p> <p><b>Implementation :</b></p> <p><input type="checkbox"/> Front End : Java/Perl/PHP/Python/Ruby/.net</p> <p><input type="checkbox"/> Backend : MongoDB/MYSQL/Oracle</p> <p><input type="checkbox"/> Database Connectivity : ODBC/JDBC</p> <p><b>Testing : Data Validation</b></p> <p>Group of students should submit the Project Report which will be consist of documentation related to different phases of Software Development Life Cycle: Title of the Project, Abstract, Introduction, scope, Requirements, Data Modeling features, Data Dictionary, Relational Database Design, Database Normalization, Graphical User Interface, Source Code, Testing document, Conclusion. Instructor should maintain progress report of mini project throughout the semester from project group and assign marks as a part of the term work</p>

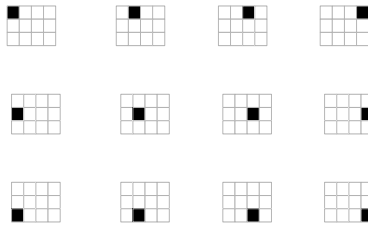
## How to Use This Manual

This Manual assumes that the facilitators are aware of Collaborative Learning Methodologies.

This Manual will only provide them tool they may need to facilitate the session on Computer Organization module in collaborative learning environment.

The Facilitator is expected to refer this Manual before the session.

<b>K</b> Applying Knowledge (PO:a)	<b>A</b> Problem Analysis (PO:b)	<b>D</b> Design & Development (PO:c)	<b>I</b> Investigation of problems (PO:d)
<b>M</b> Modern Tool Usage (PO:e)	<b>E</b> Engineer & Society (PO:f)	<b>E</b> Environment Sustainability (PO:h)	<b>T</b> Ethics (PO:i)
<b>T</b> Individual & Team work (PO:g)	<b>O</b> Communicati on (PO:k)	<b>M</b> Project Management & Finance (PO:j)	<b>I</b> Life Long Learning (PO:l)



## Disk Approach- Digital Blooms Taxonomy


















- 1: Remembering / Knowledge
- 2: Comprehension / Understanding
- 3: Applying
- 4: Analyzing
- 5: Evaluating
- 6: Creating / Design



This Manual uses icons as visual cues to the interactivities during the session.

Icons	Graduate Attributes
	Applying Knowledge
	Problem Analysis
	Design and Development
	Investigation of Problem
	Modern Tool Usage
	Engineer and Society
	Environment Sustainability
	Ethics

	Individual and Teamwork
	Communication
	Project Management and Finance
	Lifelong Learning
	<b>Blooms Taxonomy</b>
	Remembering
	Understanding
	Applying
	Analyzing
	Evaluating
	Creating

	This icon is used to indicate instructions for faculties.
	This icon is used to indicate a statement to be made by faculty.
	This icon is used to indicate a list of additional resources.
	This icon indicates an activity to be conducted.
	This icon indicates questions to be asked by faculty.

### **Program Education Outcome**

**Student will be able:-**

1	To prepare globally competent graduates having strong fundamentals, domain knowledge, updated with modern technology to provide the effective solutions for engineering problems.
2	To prepare the graduates to work as a committed professional with strong professional ethics and values, sense of responsibilities, understanding of legal, safety, health, societal, cultural and environmental issues.
3	To prepare committed and motivated graduates with research attitude, lifelong learning, investigative approach, and multidisciplinary thinking.
4	To prepare the graduates with strong managerial and communication skills to work effectively as individual as well as in teams.

### **Program Outcome**

**Student will be able:-**

1	To apply knowledge of mathematics, science, engineering fundamentals, problem solving skills, algorithmic analysis and mathematical modeling to the solution of complex engineering problems.
2	To analyze the problem by finding its domain and applying domain specific skills
3	To understand the design issues of the product/software and develop effective solutions with appropriate consideration for public health and safety, and cultural, societal, and environmental considerations.
4	To find solutions of complex problems by conducting investigations applying suitable techniques.
5	To adapt the usage of modern tools and recent software.
6	To contribute towards the society by understanding the impact of Engineering on global aspect.
7	To understand environment issues and design a sustainable system.
8	To understand and follow professional ethics.
9	To function effectively as an individual and as member or leader in diverse teams and interdisciplinary settings.
10	To demonstrate effective communication at various levels.
11	To apply the knowledge of Computer Engineering for development of projects, finance and management.
12	To keep in touch with current technologies and inculcate the practice of lifelong learning.



## Course Outcome (Modified in 18-19 SEM-I)

Student will be able to:

<b>C 304.1:</b>	Design E-R Model for given requirements and convert the same into database tables.
<b>C 304.2:</b>	Construct queries by making use of database techniques such as SQL & PL/SQL.
<b>C 304.3:</b>	Make use of modern database techniques such as NOSQL
<b>C 304.4:</b>	Explain transaction Management in relational database System.
<b>C 304.5:</b>	Analyze the use of appropriate architecture in real time environment.
<b>C 304.6:</b>	Implement advanced database Programming concepts like Big Data using HADOOP

## Experiment Learning Outcome

Student will be able to:-

ELO1	Understand and use the concept of relational databases (MySQL), in design of databases.
ELO2	Demonstrate the use of SQL objects to design and develop database using DDL statements.
ELO3	Use of SQL DML statements to design and develop for suitable database application
ELO4	Understand and use the concept of relational databases using PL/SQL, in design and Develop application.
ELO5	Understand and use the concept of NOSQL databases, in design of Databases.
ELO6	Use of NOSQL basic operation in design and develop for suitable NOSQL database application.
ELO7	Create and use of JSON objects for database application programs.
ELO8	Use the concepts of DBMS (SQL, PL/SQL and NOSQL) in design and implementation of application program

## Assignment 1

### Title: Study of Open Source Relational Databases : MySQL

**Objective:** Understanding the open source tool Mysql and its installation on fedora.

#### 1) What is database?

Database is a systematic collection of interrelated data. Databases support storage and manipulation of data. Databases make data management easy.

Examples:-

1. An online telephone directory would definitely use database to store data pertaining to people, phone numbers, other contact details, etc.
2. Your electricity service provider is obviously using a database to manage billing , client related issues, to handle fault data, etc.
3. Teacher keeps student data base like name,roll\_no,add,Date of birth.

#### 2) What is database management system?

A *database management system* (DBMS) is a collection of [programs](#) that enables you to [store](#), modify, and extract information from a [database](#).

A **DBMS** is a software that allows creation, definition and manipulation of database. Dbms is actually a tool used to perform any kind of operation on data in database.

A DBMS receives instruction from a database administrator (DBA) and accordingly instructs the system to make the necessary changes. These commands can be to load, retrieve or modify existing data from the system.

Examples:-

```
alter table employee modify column e_id int primary key;
```

#### 3) Define RDBMS?

A **relational database management system (RDBMS)** is a [database management system](#) (DBMS) that is based on the [relational model](#) . It organizes data into related rows and columns. The relational model has relationship between tables using primary keys, foreign keys and indexes.

**Features:**

- Provides data to be stored in tables
- Persists data in the form of rows and columns
- Provides facility primary key, to uniquely identify the rows
- Creates indexes for quicker data retrieval
- Provides a virtual table creation in which sensitive data can be stored & query can be applied.
- Sharing a common column in two or more tables(primary key and foreign key)
- Provides multiuser accessibility that can be controlled by individual users.

#### 4) Difference between RDBMS and DBMS?

No.	DBMS	RDBMS
1)	DBMS applications store <b>data as file</b> .	RDBMS applications store <b>data in a tabular form</b> .
2)	In DBMS, data is generally stored in either a hierarchical form or a navigational form.	In RDBMS, the tables have an identifier called primary key and the data values are stored in the form of tables.
3)	<b>Normalization is not</b> present in DBMS.	<b>Normalization is</b> present in RDBMS.
4)	DBMS does <b>not apply any security</b> with regards to data manipulation.	RDBMS <b>defines the integrity constraint</b> for the purpose of ACID (Atomicity, Consistency, Isolation and Durability) property.
5)	DBMS uses file system to store data, so there will be <b>no relation between the tables</b> .	In RDBMS, data values are stored in the form of tables, so <b>relationship</b> between these data values will be stored in the form of a table as well.
6)	DBMS has to provide some uniform methods to access the stored information.	RDBMS system supports a tabular structure of the data and relationship between them to access the stored information.
7)	DBMS <b>does not support distributed database</b> .	RDBMS <b>supports distributed database</b> .
8)	DBMS is meant to be for small organization and <b>deal with small data</b> . it supports <b>single user</b> .	RDBMS is designed to <b>handle large amount of data</b> . it supports <b>multiple users</b> .
9)	Examples of DBMS are file systems, <b>xml</b> etc.	Example of RDBMS are <b>mysql, postgres, sql server, oracle</b> etc.

#### 5) Define Table and Database?

##### Table:-

A **table** is a collection of data elements organised in terms of rows and columns. A table is also considered as a convenient representation of **relations**. But a table can have duplicate tuples while a true **relation** cannot have duplicate tuples. Table is the most simplest form of data storage. Below is an example of Employee table.

ID	Name	Age	Salary
1	Dipika	22	13000
2	Sonu	20	15000
3	Sneha	21	18000
4	Priya	19	19020

##### Database:-

Database is a systematic collection of interrelated data. Databases support storage and manipulation of data. Databases make data management easy.

##### Examples:-

1. An online telephone directory would definitely use database to store data pertaining to people, phone numbers, other contact details, etc.
2. Your electricity service provider is obviously using a database to manage billing, client related issues, to handle fault data, etc.

## 6) Define Record, Attribute?

### Record:-

A single entry in a table is called a **Record** or **Row**. A **Record** in a table represents set of related data. For example, the above **Employee** table has 4 records.

Following is an example of single record.

Gray highlighted data is called as Touple.

Yellow highlighted data is called as Attributes

1	Sonu	34	13000
2	monu	45	15000

### Attribute:-

It is the name of the column. An attribute gives the characteristics of the entity.

For example, A customer of bank may be described by: name, address, customer ID number. It is also called as data element, data field, a field, a data item, or an elementary item.

Type of Attributes in DBMS –

- 1) Simple Attribute.
- 2) Complex Attribute.
- 3) Multivalued Attribute.
- 4) Derived Attribute.
- 5) Single Attribute.

## 7) What is MY SQL?

MySQL is the world's most popular open source database. With its proven performance, reliability and ease-of-use, MySQL has become the leading database choice for web-based applications, used by high profile web properties including Facebook, Twitter, YouTube, Yahoo! and many more.

MySQL is a very powerful program in its own right. It handles a large subset of the functionality of the most expensive and powerful database packages. MySQL uses a standard form of the well-known SQL data language.

MySQL works on many operating systems and with many languages including PHP, PERL, C, C++, JAVA, etc. MySQL works very quickly and works well even with large data sets.

## 8) Steps to install MY SQL on fedora?

### 1. switch user:

```
[root@localhost admin]# yum install mysql -server
```

### 2. To install my sql type following command on terminal

```
[root@localhost admin]# systemctl start mysqld.service  
[root@localhost admin]#mysql
```

### 3. After installation is finished, check the installation

```
root@localhost admin]# mysql
```

## ASSIGNMENT 2

**Title :-** Design and Develop SQL DDL statements which demonstrate the use of SQL objects such as Table, View, Index, Sequence, Synonym.

**Objective :-** To understand the concept of DDL Operations.

### **Theroy:-**

#### **What is MY SQL?**

MySQL is the world's most popular open source database. With its proven performance, reliability and ease-of-use, MySQL has become the leading database choice for web-based applications, used by high profile web properties including Facebook, Twitter, YouTube, Yahoo! and many more.

MySQL is a very powerful program in its own right. It handles a large subset of the functionality of the most expensive and powerful database packages. MySQL uses a standard form of the well-known SQL data language.

MySQL works on many operating systems and with many languages including PHP, PERL, C, C++, JAVA, etc. MySQL works very quickly and works well even with large data sets.

### **There is three types of commands are used in database that is :-**

- 1) DDL (Data Definition Language)
- 2) DML (Data Manipulation Language)
- 3) DCL (Data Control Language)

- 1) **DDL :-**
  - i) It Is Stands For Data Definition Language
  - ii) It is used for manipulate the data.
  - iii) DDL statements are used to define the database structure or schema.

#### **Operations performs on DDL are:-**

### **1. Create operation**

- a) **Create database :-** The CREATE DATABASE statement is used to create a new SQL database.

Syntax:- CREATE DATABASE *databasename*;

- b) **Create Table :-** It is used for To Create a Table.

Syntax:- CREATE TABLE *table\_name* ( *column1 datatype*, *column2 datatype*, *column3 datatype*, .....);

2. **Alter Table :-** The ALTER TABLE statement is used to add, delete, or modify columns in an existing table

The ALTER TABLE statement is also used to add and drop various constraints on an existing table and alters the structure of the database

### **ALTER TABLE - ADD Column**

To add a column in a table, use the following syntax :-

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

### **ALTER TABLE - DROP COLUMN**

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

```
Syntax:- ALTER TABLE table_name  
DROP COLUMN column_name;
```

### **ALTER TABLE - ALTER/MODIFY COLUMN**

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

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

- 1) **Drop Table :-** The DROP TABLE statement is used to drop an existing table in a database.

```
Syntax:-DROP TABLE table_name;
```

- 2) **Truncate Table :-** The TRUNCATE TABLE statement is used to delete the data inside a table, but not the table itself.

```
Syntax:-TRUNCATE TABLE table_name ;
```

- 3) **Rename Table :-** It is used to rename an object . It is used for give another name to the table.

```
Syntax :- Rename old_table name to New_table name ;
```

## **SQL Objects :-**

- 1) **Table**

A table is a collection of related data held in a structure format within database it consists of column and row a table is a set of data elements using a model of vertical column and horizontal rows the cell being the init where a row and column insert .

## 2) View

In SQL, a view is a virtual table based on the result-set of an SQL statement. A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

You can add SQL functions, WHERE, and JOIN statements to a view and present the data as if the data were coming from one single table.

CREATE VIEW

**Syntax :- CREATE VIEW view\_name AS SELECT column1, column2, ...  
FROM table\_name WHERE condition;**

### a) SQL CREATE OR REPLACE VIEW Syntax

**CREATE OR REPLACE VIEW view\_name AS  
SELECT column1, column2, ...  
FROM table\_name  
WHERE condition;**

### b) SQL Dropping a View

You can delete a view with the DROP VIEW command.

**DROP View Syntax:-DROP VIEW view\_name;**

## 3) 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.

### a) **CREATE INDEX :-** Creates an index on a table. Duplicate values are allowed:

**Syntax:- CREATE INDEX index\_name  
ON table\_name (column1, column2, ...);**

### b) **CREATE UNIQUE INDEX :-**Creates a unique index on a table. Duplicate values are not allowed:

**Syntax:- CREATE UNIQUE INDEX index\_name  
ON table\_name (column1, column2, ...);**



#### 4) **Sequence :-**

Auto-increment allows a unique number to be generated automatically when a new record is inserted into a table.

Often this is the primary key **field** that we would like to be created automatically every time a new record is inserted.

**Syntax :-** Create table < table\_name > (variable\_name datatype primary key auto increment , variable\_name data type);

**Conclusion :-** Here we understood the DDL command and SQL object like, TABLE, VIEW INDEX and SEQUENCES operations using DDL commands .

## Assignment No:3

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

**Objective:** Understand the concept of DML Commands and its operations with Operators, functions.

### Theory:

#### What is SQL?

SQL stands for Structured Query Language. SQL is used to communicate with a database. According to ANSI (American National Standards Institute), it is the standard language for relational database management systems. SQL statements are used to perform tasks such as update data on a database, or retrieve data from a database. Some common relational database management systems that use SQL are: Oracle, Sybase, Microsoft SQL Server, Access, Ingres, etc. Although most database systems use SQL, most of them also have their own additional proprietary extensions that are usually only used on their system. However, the standard SQL commands such as "Select", "Insert", "Update", "Delete", "Create", and "Drop" can be used to accomplish almost everything that one needs to do with a database.

#### SQL Languages:

- 1]DDL (Data Definition Language).
- 2]DML (Data Manipulation Languages).
- 3]DCL (Data Control Languages).

#### What is DML ?

A popular data manipulation language is that of [Structured Query Language](#) (SQL), which is used to retrieve and manipulate [data](#) in a [relational database](#). it is used to manipulate **data itself**. Data manipulation language comprises the SQL data change statements, which modify stored data but not the schema or database objects. For example, with SQL, it would be instructions such as **insert**, **update**, **delete**, **select**.

- **Insert:** The INSERT statement is used to insert new records in a table.

**Syntax:** It is possible to write the INSERT statement in two ways. The first way specifies both the column names and the values to be inserted:

**INSERT INTO table\_name (column1, column2, column3, ...)  
VALUES (value1, value2, value3, ...);**

If you are adding values for all the columns of the table, you do not need to specify the column names in the SQL query. However, make sure the order of the values is in the same order as the columns in the table. The INSERT syntax would be as follows:

**INSERT INTO table\_name  
VALUES (value1, value2, value3, ...);**

- **Select:**

The SELECT statement is used to select data from a database. The data returned is stored in a result table, called the result-set.

**Syntax:**

**SELECT column1, column2, ...  
FROM table\_name;**

Here, column1, column2, ... are the field names of the table you want to select data from. If you want to select all the fields available in the table, use the following syntax:

**SELECT \* FROM table\_name;**

- **Update:**

The UPDATE statement is used to modify the existing records in a table.

**Syntax: UPDATE table\_name**

**SET column1 = value1, column2 = value2, ...  
WHERE condition;**

- **Delete:**

The DELETE statement is used to delete existing records in a table.

**Syntax:**

**DELETE FROM table\_name  
WHERE condition;**

### **What is Functions?**

A user-defined function is a Transact-SQL or common language runtime (CLR) routine that accepts parameters, performs an action, such as a complex calculation, and returns the result of that action as a value. The return value can either be a scalar (single) value or a table. SQL Server has many built-in functions. This reference contains the string, numeric, date, conversion, and advanced.

The SQL Functions are: MIN(),MAX(),COUNT(),AVG(),SUM() and ORDERBY().

- **MIN():**The MIN() function returns the smallest value of the selected column.

**Syntax:**

**SELECT MIN(column\_name)  
FROM table\_name  
WHERE condition;**

- **MAX():**The MAX() function returns the largest value of the selected column.

**Syntax:**

**SELECT MAX(column\_name)**  
**FROM table\_name**  
**WHERE condition;**

- **COUNT():**The COUNT() function returns the number of rows that matches a specified criteria.

**Syntax:**

**SELECT COUNT(column\_name)**  
**FROM table\_name**  
**WHERE condition;**

- **AVG():**The AVG() function returns the average value of a numeric column.

**Syntax:**

**SELECT AVG(column\_name)**  
**FROM table\_name**  
**WHERE condition;**

- **SUM():**The SUM() function returns the total sum of a numeric column.

**Syntax:**

**SELECT SUM(column\_name)**  
**FROM table\_name**  
**WHERE condition;**

- **ORDERBY():** The ORDER BY keyword is used to sort the result-set in ascending or descending order. The ORDER BY keyword sorts the records in ascending order by default. To sort the records in descending order, use the DESC keyword.

**Syntax:**

**SELECT column1, column2, ...**  
**FROM table\_name**  
**ORDER BY column1, column2, ... ASC|DESC;**

### **What is operators?**

An **operator** is a reserved word or a character used primarily in an **SQL** statement's **WHERE** clause to perform operation(s), such as comparisons and arithmetic operations. These **Operators** are

used to specify conditions in an **SQL** statement and to serve as conjunctions for multiple conditions in a statement.

The Operators are: **AND, OR, NOT, BETWEEN, LESS THAN, GREATER THAN.**

- **AND:** The WHERE clause can be combined with AND operators. The AND operators are used to filter records based on more than one condition. The AND operator displays a record if all the conditions separated by AND is TRUE.

**Syntax:**

```
SELECT column1, column2, ...  
FROM table_name  
WHERE condition1 AND condition2 AND condition3 ...;
```

- **OR:** The WHERE clause can be combined with OR operators. The OR operators are used to filter records based on more than one condition: The OR operator displays a record if any of the conditions separated by OR is TRUE.

**Syntax:**

```
SELECT column1, column2, ...  
FROM table_name  
WHERE condition1 OR condition2 OR condition3 ...;
```

- **NOT:** The WHERE clause can be combined with AND, OR, and NOT operators. The NOT operator displays a record if the condition(s) is NOT TRUE.

**Syntax:**

```
SELECT column1, column2, ...  
FROM table_name  
WHERE NOT condition;
```

- **BETWEEN:** The BETWEEN operator selects values within a given range. The values can be numbers, text, or dates. The BETWEEN operator is inclusive: begin and end values are included.

**Syntax:**

```
SELECT column_name(s)  
FROM table_name  
WHERE column_name BETWEEN value1 AND value2;
```

- **LESS THAN:** Less than operator is used to find out less value from table. We can use WHERE clause.

**Syntax:**

**SELECT \* from table\_name WHERE condition;**

- **GREATER THAN:** Greater than operator is used to find out greater value from table. We can use WHERE clause.

**Syntax:**

**SELECT \* from table\_name WHERE condition;**

**Conclusion:** We have studied SQL DML commands and SQL Functions and Operators and perform all the queries on database table.

## Assignment No-4

**Title:-** To design SQL queries using join operation.

**Objective:-** Understand the concept of SQL JOIN operation and its types, to Design SQL queries using join operation .

**Theory:-**

**Join:-**

The SQL **Joins** clause is used to combine records from two or more tables in a database. A JOIN is a means for combining fields from two tables by using values common to each. Consider the following two tables

**Types of join:-**

- INNER JOIN
- OUTER JOIN
  - Left outer join
  - Right outer join
  - full outer join
- NATURAL JOIN
- CROSS JOIN or CARTESION JOIN

### 1.) INNER JOIN:-

This is a simple JOIN in which the result is based on matched data as per the equality condition specified in the query.

**Inner Join Syntax is:-**

```
SELECT column-name-list  
from table-name1  
INNER JOIN  
table-name2  
WHERE table-name1.column-name = table-name2.column-name;
```

**Inner JOIN** query will be,

```
SELECT * from class, class_info where class.id = class_info.id;
```

### 2.) Natural JOIN:-

Natural Join is a type of Inner join which is based on column having same name and same datatype present in both the tables to be joined.

Natural Join Syntax is:-

```
SELECT *
```

**from *table-name1***  
**NATURAL JOIN**  
***table-name2*;**

### **3.) Outer JOIN:-**

Outer Join is based on both matched and unmatched data. Outer Joins subdivide further into,

- Left Outer Join
- Right Outer Join
- Full Outer Join

#### **Left Outer Join:-**

The left outer join returns a result table with the **matched data** of two tables then remaining rows of the **left** table and null for the **right** table's column.

Left Outer Join syntax is,

**SELECT column-name-list**  
**from *table-name1***  
**LEFT OUTER JOIN**  
***table-name2***  
**on table-name1.column-name = table-name2.column-name;**

Left outer Join Syntax for **Oracle** is,

**select column-name-list**  
**from *table-name1*,**  
***table-name2***  
**on table-name1.column-name = table-name2.column-name(+);**

#### **Right Outer Join:-**

The right outer join returns a result table with the **matched data** of two tables then remaining rows of the **right table** and null for the **left** table's columns.

**Right Outer Join Syntax is,**

**select column-name-list**  
**from *table-name1***  
**RIGHT OUTER JOIN**  
***table-name2***  
**on table-name1.column-name = table-name2.column-name;**

Right outer Join Syntax for **Oracle** is,

**select column-name-list**  
**from *table-name1*,**  
***table-name2***  
**on table-name1.column-name(+) = table-name2.column-name;**



### ***Full Outer Join:-***

The full outer join returns a result table with the **matched data** of two table then remaining rows of both **left** table and then the **right** table.

**Full Outer Join Syntax is,**

```
select column-name-list  
from table-name1  
FULL OUTER JOIN  
table-name2  
on table-name1.column-name = table-name2.column-name;
```

### **4.) CROSS JOIN/CARTESION JOIN:-**

The CARTESIAN JOIN or CROSS JOIN returns the Cartesian product of the sets of records from two or more joined tables. Thus, it equates to an inner join where the join-condition always evaluates to either True or where the join-condition is absent from the statement.

#### **Syntax**

The basic syntax of the **CARTESIAN JOIN** or the **CROSS JOIN** is as follows –

```
SELECT table1.column1, table2.column2...  
FROM table1, table2 [, table3 ]
```

**Conclusion:** Here we understood the concept of SQL join operation and its types. Learned how to design sub queries using join.

## Case study on PLSQL

### Q1. WHAT IS PL/SQL?

**ANS:** In Oracle database management, PL/SQL is a procedural language extension to Structured Query Language (SQL). The purpose of PL/SQL is to combine database language and procedural programming language.

### Q2. FEATURES OF PLSQL?

**ANS:** PL/SQL has the following features –

- PL/SQL is tightly integrated with SQL.
- It offers extensive error checking.
- It offers numerous data types.
- It offers a variety of programming structures.
- It supports structured programming through functions and procedures.
- It supports object-oriented programming.
- It supports the development of web applications and server pages.

### Q3.DIFFERENCE BETWEEN SQL AND PL/SQL?

**ANS:**

S. No.	SQL	PL/SQL
1.	SQL Stands for Structured Query Language	PL/SQL stands for Programming Language SQL.
2.	SQL is used to execute single query or statement at a time.	PL/SQL is used to execute block of code or program that have multiple statements.
3.	SQL tells the database what to do but not how to do. So it is declarative.	PL/SQL tells the database how to do things. So it is procedural.
4.	SQL can be used in PL/SQL programs.	PL/SQL can't be used in SQL statement.
5.	SQL is used with various database systems like MySQL, SQL Server, Oracle, DB2, etc.	PL/SQL is used only with Oracle database.
6.	An example of SQL query is given below.  SELECT * FROM Customers;	An example of PL/SQL program is given below.  BEGIN  dbms_output.put_line('Hello Workd');  END;/

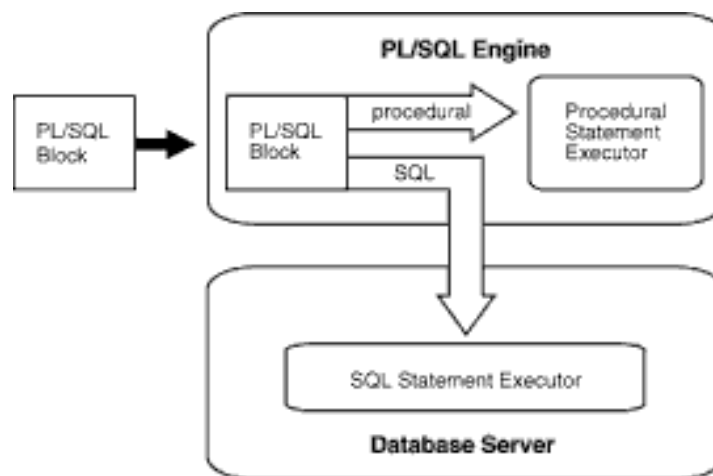
### Q4.WHAT ARE THE ADVANTAGES OF PL/SQL?

**ANS:** Advantages of PL/SQL are as follows:

1. PL/SQL is structured as it consists of blocks of code and hence streamlined. This makes PL/SQL highly productive.
2. It is highly portable, has immense error handling mechanisms.
3. High performance as lines of code can be sent to oracle. This reduces traffic.
4. With the user of stored procedures, PL/SQL is highly secured.
5. Extremely flexible and easy to learn with syntaxes like SELECT, INSERT, UPDATE etc.
6. Support SQL data manipulation.
7. Provide facilities like conditional checking, branching and looping.
8. Provide fast code execution since it sends SQL statement as a block to the oracle engine.

#### Q5. ARCHITECTURES OF PL/SQL?

ANS:



#### PL/SQL Engine:

The PL/SQL compilation and run-time system is an engine that compiles and runs PL/SQL units. The engine can be installed in the database or in an application development tool, such as Oracle Forms.

In either environment, the PL/SQL engine accepts as input any valid PL/SQL unit. The engine runs procedural statements, but sends SQL statements to the SQL engine in the database, as shown in [figure](#).

Typically, the database processes PL/SQL units.

When an application development tool processes PL/SQL units, it passes them to its local PL/SQL engine. If a PL/SQL unit contains no SQL statements, the local engine processes the entire PL/SQL unit. This is useful if the application development tool can benefit from conditional and iterative control.

For example, Oracle Forms applications frequently use SQL statements to test the values of field entries and do simple computations. By using PL/SQL instead of SQL, these applications can avoid calls to the database.

#### PL/SQL Units and Compilation Parameters

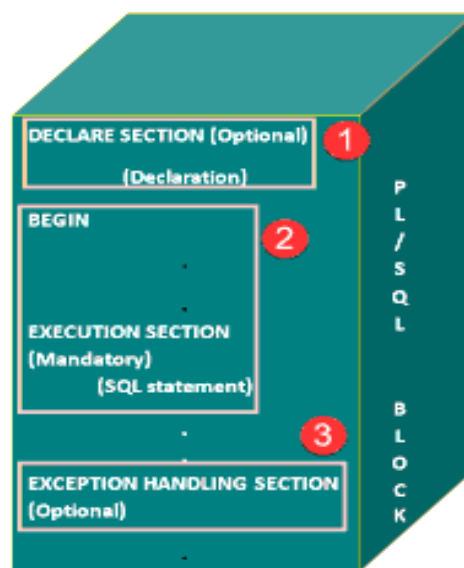
A PL/SQL unit is one of these:

- PL/SQL anonymous block

- FUNCTION
- LIBRARY
- PACKAGE
- PACKAGE BODY
- PROCEDURE
- TRIGGER
- TYPE
- TYPE BODY

#### Q6.BLOCK STRUCTURE OF PL/SQL?

ANS:



#### Block Structure

PL/SQL blocks have a pre-defined structure in which the code is to be grouped. Below are different sections of PL/SQL blocks

- Declaration section
- Execution section
- Exception-Handling section

The below picture illustrates the different PL/SQL block and their section order.

#### Declaration Section

This is the first section of the PL/SQL blocks. This section is an optional part. This is the section in which the declaration of variables, cursors, exceptions, subprograms, pragma instructions and collections that are needed in the block will be declared. Below are few more characteristics of this part.

- This particular section is optional and can be skipped if no declarations are needed.
- This should be the first section in a PL/SQL block, if present.

- This section starts with the keyword 'DECLARE' for triggers and anonymous block. For other subprograms this keyword will not be present, instead the part after the subprogram name definition marks the declaration section.
- This section should be always followed by execution section.

### Execution Section

Execution part is the main and mandatory part which actually executes the code that is written inside it. Since the PL/SQL expects the executable statements from this block this cannot be an empty block, i.e., it should have at least one valid executable code line in it. Below are few more characteristics of this part.

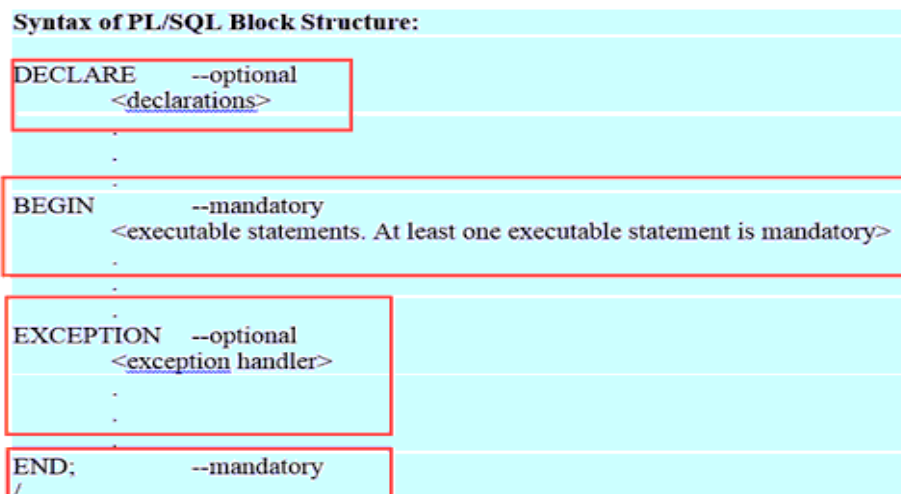
- This can contain both PL/SQL code and SQL code.
- This can contain one or many blocks inside it as a nested blocks.
- This section starts with the keyword 'BEGIN'.
- This section should be followed either by 'END' or Exception-Handling section (if present)

### Exception-Handling Section:

The exception are unavoidable in the program which occurs at run-time and to handle this Oracle has provided an Exception-handling section in blocks. This section can also contain PL/SQL statements. This is an optional section of the PL/SQL blocks.

- This is the section where the exception raised in the execution block is handled.
- This section is the last part of the PL/SQL block.
- Control from this section can never return to the execution block.
- This section starts with the keyword 'EXCEPTION'.
- This section should be always followed by the keyword 'END'.

The Keyword 'END' marks the end of PL/SQL block. Below is the syntax of the PL/SQL block structure.



### Q7. WRITE INSTALLATION PROCEDURE OF PL/SQL ON WINDOWS OR FEDORA:

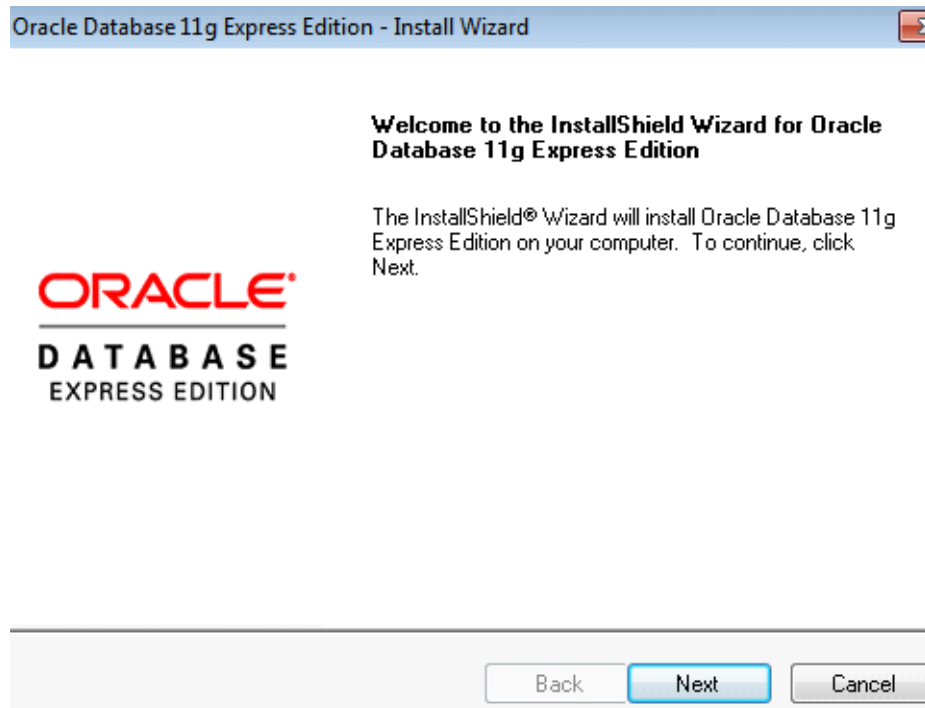
**ANS: Installation Steps of PL/SQL on windows7/8/10.**

1.Download **Oracle 11g Express Edition** and **SQL Developer** software from oracle.com from official site.

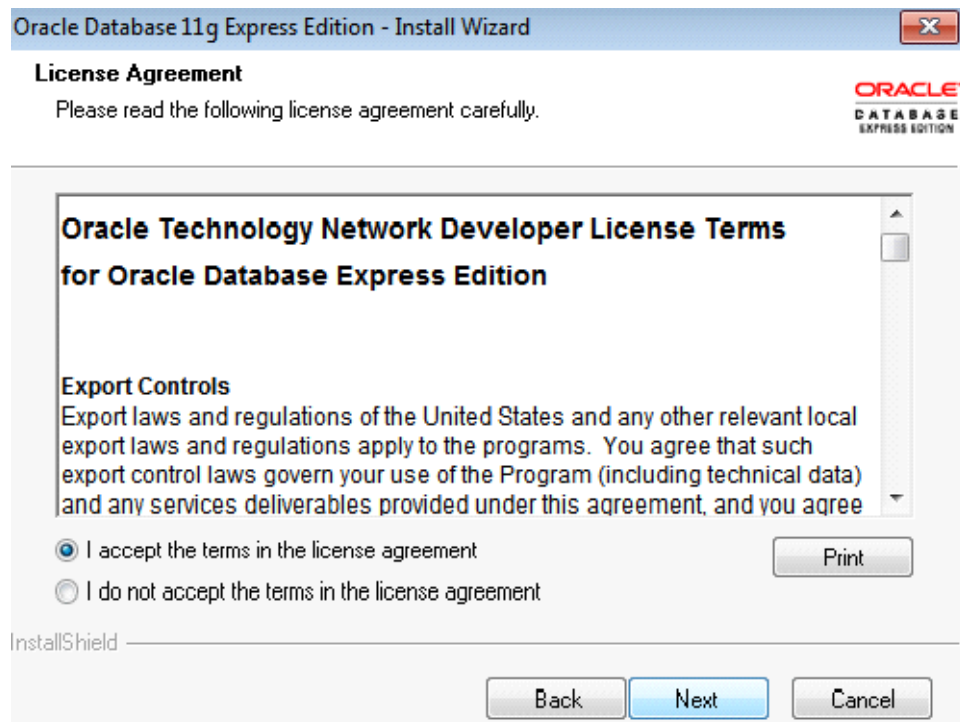
2.Extract the .rar files of **Oracle 11g Express Edition** and **SQL** at specified location of your system..

3. Open DISK1 folder which is extracted file of **Oracle 11g Express Edition** .  
Then right click on setup icon , run as administrator.

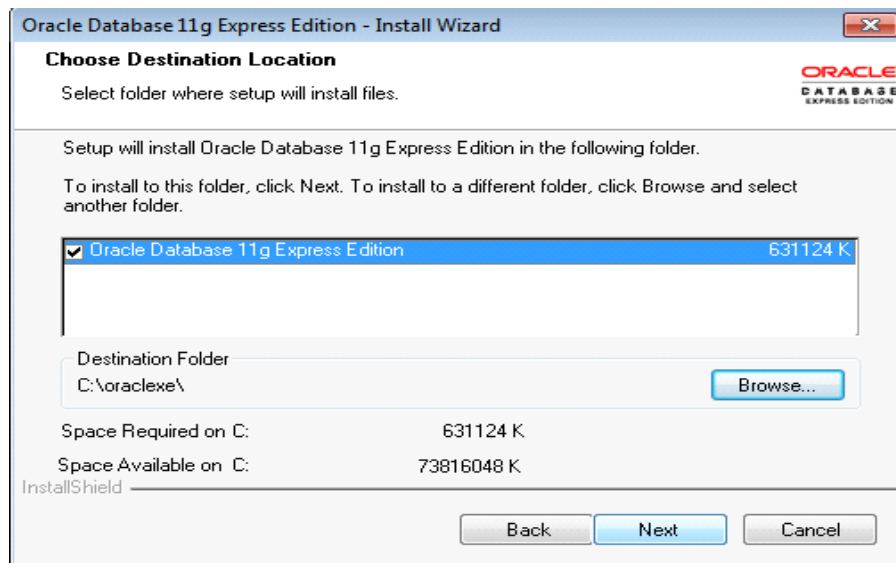
4. It will start installing click on next.



5. Then click on “I accept the terms in the license agreement”. Then click on next.

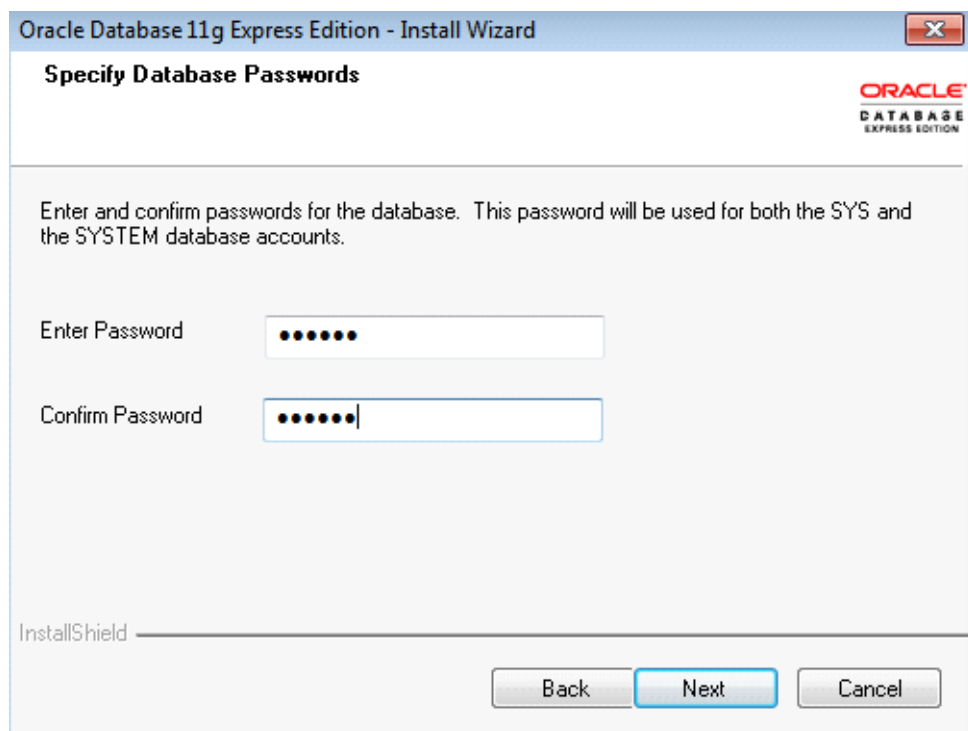


6. Specify the installation location .

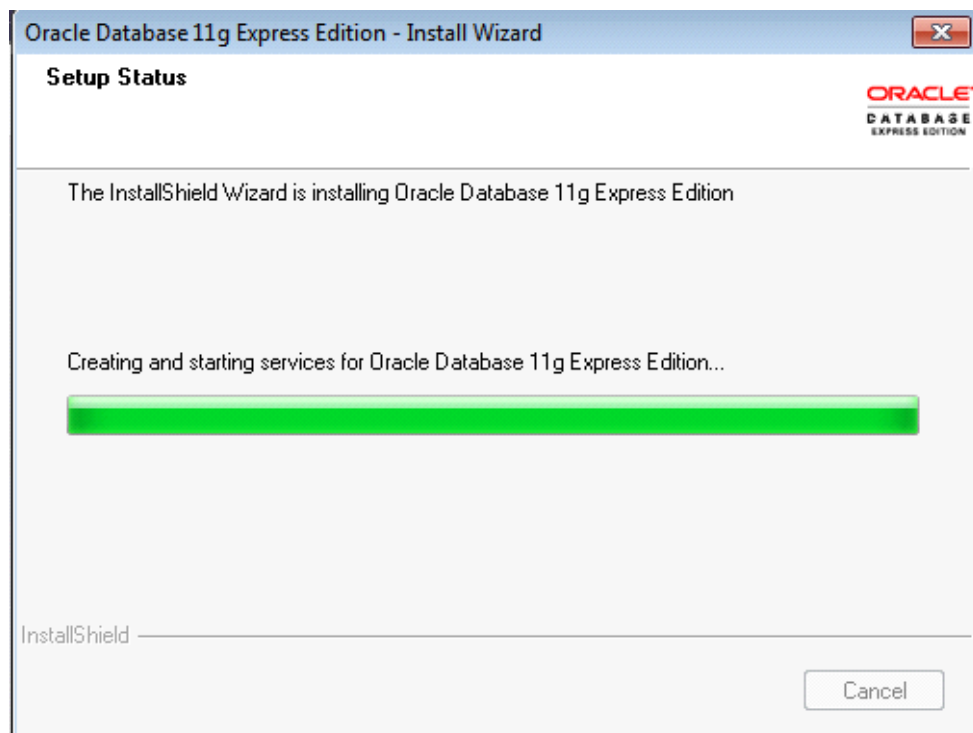
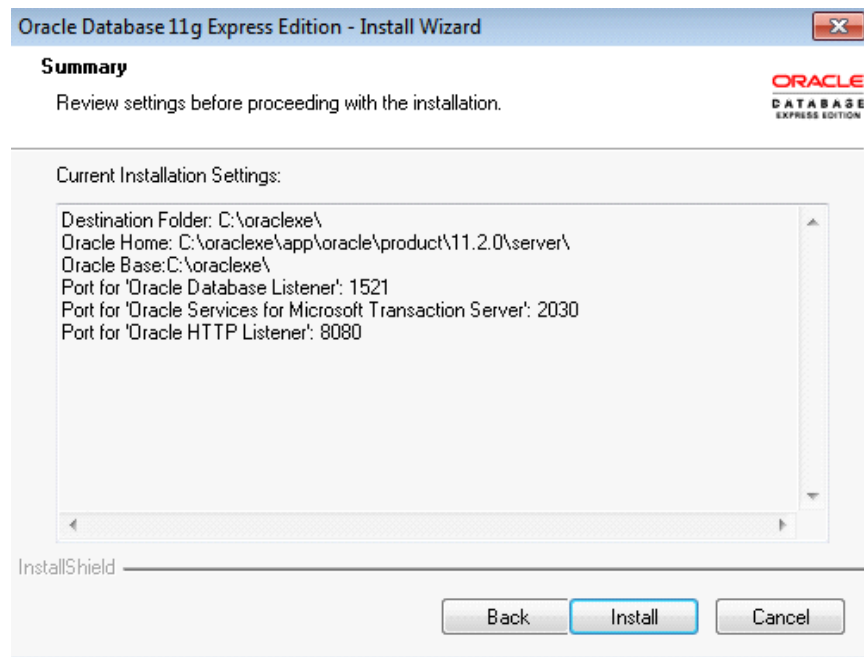


**7.Enter password here.**

**Password:system.**



**8.Click on next and install.**



**9. Click on Finish.**



Oracle Database 11g Express Edition - Install Wizard



**InstallShield Wizard Complete**

Setup has finished installing Oracle Database 11g Express Edition on your computer.



Back

Finish

Cancel

## Assignment No:-5

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

Schema:

1. Borrower(Roll\_no, Name, DateofIssue, NameofBook, Status)

2. Fine(Roll\_no,Date,Amt)

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

**Objective:-**To understand the of Control structure and Exception handling of pl/sql block structure.

**Theroy:-**

**Q. 1.What is an anonymous/ Unnamed block in PL SQL?**

ANS:-

The PL/SQL anonymous block statement is an executable statement that can contain **PL/SQL** control statements and **SQL** statements. It can be used to implement procedural logic in a scripting language. ... The exception section must begin with the keyword **EXCEPTION**, and continues until the end of the **block** in which it appears.

### Unnamed block statement (PL/SQL)

The PL/SQL anonymous block statement is an executable statement that can contain PL/SQL control statements and SQL statements. It can be used to implement procedural logic in a scripting language. In PL/SQL contexts, this statement can be compiled and executed by the DB2® data server.

The anonymous block statement, which does not persist in the database, can consist of up to three sections: an optional declaration section, a mandatory executable section, and an optional exception section.

The optional declaration section, which can contain the declaration of variables, cursors, and types that are to be used by statements within the executable and exception sections, is inserted before the executable BEGIN-END block.

The optional exception section can be inserted near the end of the BEGIN-END block. The exception section must begin with the keyword **EXCEPTION**, and continues until the end of the block in which it appears.

**Description:-**

**DECLARE:-**

An optional keyword that starts the DECLARE statement, which can be used to declare data types, variables, or cursors. The use of this keyword depends upon the context in which the block appears.

declaration

Specifies a variable, cursor, or type declaration whose scope is local to the block. Each declaration must be terminated by a semicolon.

**BEGIN:-**

A mandatory keyword that introduces the executable section, which can include one or more SQL or PL/SQL statements. A BEGIN-END block can contain nested BEGIN-END blocks.

statement

Specifies a PL/SQL or SQL statement. Each statement must be terminated by a semicolon.

**EXCEPTION:-**

An optional keyword that introduces the exception section.

WHEN exception-condition

Specifies a conditional expression that tests for one or more types of exceptions.

THEN handler-statement

Specifies a PL/SQL or SQL statement that is executed if a thrown exception matches an exception in *exception-condition*. Each statement must be terminated by a semicolon.

**END:-**

A mandatory keyword that ends the block.

**Design and Implementation:**

**Create borrower table:-**

Create table borrower(roll\_no number(5),name varchar2(20),Dateofissue date,NameBook varchar2(20),status varchar(20));

**Insert record in borrower table:-**

Insert into borrower values(1,'om','18-sep-2017','DBMS','I');

Insert into borrower values(2,'priya','19-sep-2017','C++','I');

Insert into borrower values(3,'omkar','20-sep-2017','java','I');

**Create fine table:**

Create table fine(roll\_no number(5),sdate date,Amt number(5));

**Write PROCEDURE PL/SQL Block:-**

DECLARE

Roll\_No NUMBER(3);

BookName varchar2(50);

IssueDate DATE;

CurrentDate DATE;

```

NoOfDays Number(2);
Amount Number;
BEGIN
DBMS_OUTPUT.PUT_LINE('Enter Student Roll Number');
Roll_No := &rollno;
DBMS_OUTPUT.PUT_LINE('Enter Book Name');
BookName := '&bookname';
CurrentDate := trunc(SYSDATE);
SELECT DateOfIssue into IssueDate FROM Borrower WHERE RollNo = Roll_No AND
NameOfBook =BookName;
SELECT trunc(SYSDATE) - IssueDate INTO NoOfDays from dual;
DBMS_OUTPUT.PUT_LINE('No of Days' || NoOfDays);
IF (NoOfDays> 30) THEN
Amount := NoOfDays * 50;
ELSIF (NoOfDays>= 15 AND NoOfDays<=30) THEN
Amount := NoOfDays * 5;
END IF;
IF Amount > 0 THEN
INSERT INTO Fine values (Roll_No, sysdate, Amount);
END IF;
UPDATE Borrower SET Status = 'R' WHERE RollNo=Roll_No;
END;

* /END OF PROCEDURE/*

```

### **Execute procedure**

**Conclusion:-** Here we understood the concept of unnamed block of PL/SQL structure. Designed the library due management application using unnamed block of PL/SQL.

## **Assignment 6**

**Title:** Cursors: (All types: Implicit, Explicit, Cursor FOR Loop, Parameterized Cursor) Write a PL/SQL block of code using parameterized Cursor, that will merge the data available in the newly created table N\_RollCall with the data available in the table O\_RollCall. If the data in the first table already exist in the second table then that data should be skipped. Frame the separate problem statement for writing PL/SQL block to implement all types of Cursors in-line with above statement. The problem statement should clearly state the requirements.

**Objective:** Learning the cursor and its type, how to define cursors on database to operate and manage the database.

**Theory:**

**Definition :**

A cursor is a temporary work area created in the system memory when a SQL statement is executed. A cursor contains information on a select statement and the rows of data accessed by it. This temporary work area is used to store the data retrieved from the database, and manipulate this data.

**Explanation :**

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 is used to referred to a program to fetch and process the rows returned by the SQL statement, one at a time.

**Types of Cursor :**

There are two types of cursors:

- 1] Implicit Cursors
- 2] Explicit Cursors

**1] Implicit Cursors :**

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

Whenever a DML statement (INSERT, UPDATE and DELETE) is issued, an implicit cursor is associated with this statement. For INSERT operations, the cursor holds the data that needs to be inserted. For UPDATE and DELETE operations, the cursor identifies the rows that would be affected.

In PL/SQL, you can refer to the most recent implicit cursor as the **SQL cursor**, which always has attributes such as **%FOUND**, **%ISOPEN**, **%NOTFOUND**, and **%ROWCOUNT**. The SQL cursor has additional attributes, **%BULK\_ROWCOUNT** and **%BULK\_EXCEPTIONS**, designed

for use with the **FORALL** statement. The following table provides the description of the most used attributes –

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

**NOTE: Any SQL cursor attribute will be accessed as sql%attribute\_name**

## **2] Explicit Cursors:**

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

The syntax for creating an explicit cursor is –

**CURSOR cursor\_name IS select\_statement;**

Working with an explicit cursor includes the following steps –

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

### **Declaring the Cursor :**

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

#### **Syntax for explicit cursor declaration :**

1. CURSOR name IS
2. SELECT statement;

### **Opening the Cursor :**

Opening the cursor allocates the memory for the cursor and makes it ready for fetching the rows returned by the SQL statement into it.

#### **Syntax for cursor open :**

1. OPEN cursor\_name;

### **Fetching the Cursor :**

Fetching the cursor involves accessing one row at a time.

#### **Syntax for cursor fetch:**

FETCH cursor\_name INTO variable\_list;

### **Closing the Cursor :**

Closing the cursor means releasing the allocated memory.

#### **Syntax for cursor close :**

Close cursor\_name;

### **Advantages of Cursor :**

- 1] Using cursor to getting multiple values.
- 2] Where Current of Clause: this is use full when the primary key is not present.
- 3] Cursors can be faster than a while loop but they do have more overhead.
- 4] we can do RowWise validation or in other way you can perform operation on each Row.

**Conclusion:** Here we understood the concept of cursor and its types and how to define and use of cursors on database table to read and fetch the data.

## Assignment No: 7

**Title:** PL/SQL Stored Procedure and Stored Function.

Write a Stored Procedure namely `proc_Grade` for the categorization of student. If marks scored by students in examination is  $\leq 1500$  and  $\geq 990$  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 for using procedure created with above requirement.

`Stud_Marks(name, total_marks)`

`Result(Roll, Name, Class)`

Frame the separate problem statement for writing PL/SQL Stored Procedure and function, inline with above statement.

**Objectives:** To understand the concept of stored procedure and stored function.

**Theory:**

**Stored procedure:**

A stored procedure is a set of Structured Query Language (SQL) statements with an assigned name, which are stored in a relational database management system as a group, so it can be reused and shared by multiple programs.

Stored procedures can access or modify data in a database, but it is not tied to a specific database or object, which offers a number of advantages.

**Benefits of using stored procedures:**

A stored procedure provides an important layer of security between the user interface and the database. It supports security through data access controls because end users may enter or change data, but do not write procedures. A stored procedure preserves data integrity because information is entered in a consistent manner. It improves productivity because statements in a stored procedure only must be written once.

**Creating a Procedure:**

A procedure is created with the **CREATE OR REPLACE PROCEDURE** statement.

**Syntax:**

```
CREATE [OR REPLACE] PROCEDURE procedure_name [(parameter_name [IN | OUT | IN OUT]
type [, ...])]
```

```
{IS | AS}
```

```
BEGIN
```

```
    < procedure_body >
```

```
END procedure_name;
```



**Where,**

- procedure-name specifies the name of the procedure.
- [OR REPLACE] option allows the modification of an existing procedure.
- The optional parameter list contains name, mode and types of the parameters. IN represents the value that will be passed from outside and OUT represents the parameter that will be used to return a value outside of the procedure.
- procedure-body contains the executable part.
- The AS keyword is used instead of the IS keyword for creating a standalone procedure.

**Stored function:**

A stored function (also called a user function or user-defined function) is a set of PL/SQL statements you can call by name. Stored functions are very similar to procedures, except that a function returns a value to the environment in which it is called. User functions can be used as part of a SQL expression.

**Creating a Function**

A standalone function is created using the **CREATE FUNCTION** statement.

**Syntax:**

```
CREATE [OR REPLACE] FUNCTION function_name [(parameter_name [IN | OUT | IN OUT]
type [, ...])]
```

```
RETURN return_datatype
```

```
{ IS | AS }
```

```
BEGIN
```

```
< function_body >
```

```
END [function_name];
```

**Where,**

- function-name specifies the name of the function.
- [OR REPLACE] option allows the modification of an existing function.
- The optional parameter list contains name, mode and types of the parameters. IN represents the value that will be passed from outside and OUT represents the parameter that will be used to return a value outside of the procedure.
- The function must contain a return statement.
- The RETURN clause specifies the data type you are going to return from the function.
- function-body contains the executable part.

-The AS keyword is used instead of the IS keyword for creating a standalone function.

### **Design And Implementation:**

**step1:**

**create a table as stud\_marks( rollno , Name ,marks);**

**step 2:**

**create a table as Result (rollno , Name, class);**

**step 3:**

**write a procedure as proc\_grade**

If marks scored by students in examination is  $\leq 1500$  and  $\text{marks} \geq 990$  then student will be placed indistinction category, if marks scored are between 989 and 900 category is first class, if marks 899 and 825 category is Higher Second Class.

**Conclusion:** Here we understood the concept of stored procedure and function that could be helpful for accessing and retrieving data efficiently from the database.

## Assignment 8

**Title:** Database Trigger (All Types: Row level and Statement level triggers, Before and After Triggers). Write a database trigger on Library table. The System should keep track of the records that are being updated or deleted. The old value of updated or deleted records should be added in Library\_Audit table. Frame the problem statement for writing Database Triggers of all types, in-line with above statement. The problem statement should clearly state the requirements.

**Objective:** Understand the concept of Triggers and use of triggers on database table.

### Theory:

#### Definition :

A trigger is defined for a specific table and one or more events. In most database management systems you can only define one trigger per table.

#### Syntax :

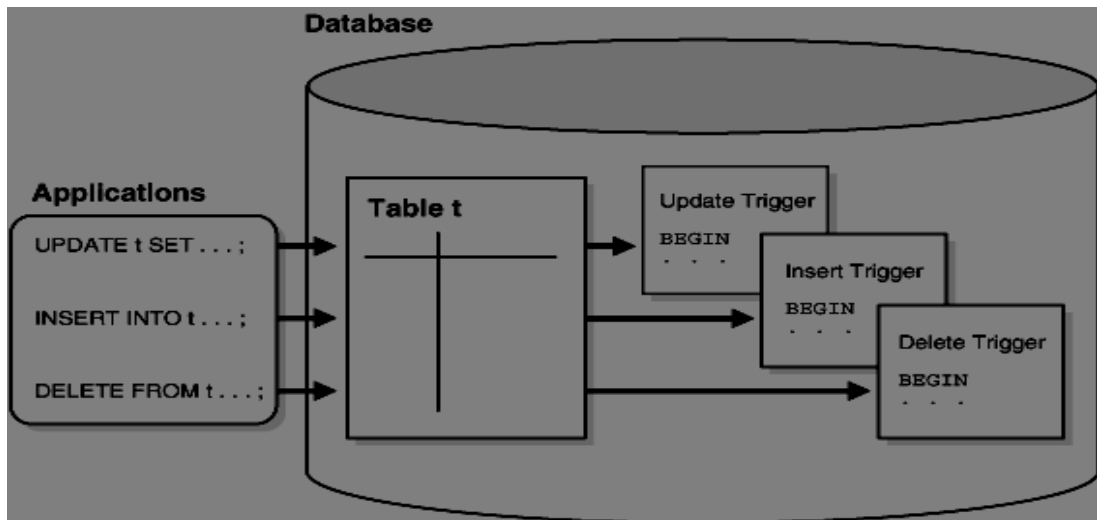
-- SQL Server Syntax --

Trigger on an INSERT, UPDATE, or DELETE statement to a table or view (DML Trigger)

```
CREATE [ OR ALTER ] TRIGGER [ schema_name . ]trigger_name
ON { table | view }
[ WITH <dml_trigger_option> [ ,...n ] ]
{ FOR | AFTER | INSTEAD OF }
{ [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }
[ WITH APPEND ]
[ NOT FOR REPLICATION ]
AS { sql_statement [ ; ] [ ,...n ] | EXTERNAL NAME <method specifier [ ; ] > }
```

```
<dml_trigger_option> ::=
    [ ENCRYPTION ]
    [ EXECUTE AS Clause ]
```

```
<method_specifier> ::=
    assembly_name.class_name.method_name
```



**Explanation :**

### What is a Database Trigger?

A database trigger is special stored procedure that is run when specific actions occur within a database. Most triggers are defined to run when changes are made to a table's data. Triggers can be defined to run instead of or after DML (Data Manipulation Language) actions such as INSERT, UPDATE, and DELETE.

Triggers help the database designer ensure certain actions, such as maintaining an audit file, are completed regardless of which program or user makes changes to the data.

The programs are called triggers since an event, such as adding a record to a table, fires their execution.

Triggers and their implementations are specific to database vendors. In this article we'll focus on Microsoft SQL server; however, the concepts are the same or similar in Oracle and MySQL.

Note: All the examples for this lesson are based on Microsoft SQL Server Management Studio and the AdventureWorks2012 database. You can get started using these free tools using my Guide Getting Started Using SQL Server.

### Events

The triggers can occur AFTER or INSTEAD OF a DML action. Triggers are associated with the database DML actions INSERT, UPDATE, and DELETE. Triggers are defined to run when these actions are executed on a specific table.

### Triggering Event or Statement

A triggering event or statement is the SQL statement that causes a trigger to be fired. A triggering event can be an INSERT, UPDATE, or DELETE statement on a table.

For example, in Figure 15 - 3, the triggering statement is

... UPDATE OF parts\_on\_hand ON inventory ...

which means that when the PARTS\_ON\_HAND column of a row in the INVENTORY table is updated, fire the trigger. Note that when the triggering event is an UPDATE statement, you can include a column list to identify which columns must be updated to fire the trigger. Because INSERT

and DELETE statements affect entire rows of information, a column list cannot be specified for these options.

A triggering event can specify multiple DML statements, as in

```
... INSERT OR UPDATE OR DELETE OF inventory ...
```

which means that when an INSERT, UPDATE, or DELETE statement is issued against the INVENTORY table, fire the trigger. When multiple types of DML statements can fire a trigger, conditional predicates can be used to detect the type of triggering statement. Therefore, a single trigger can be created that executes different code based on the type of statement that fired the trigger.

### **Trigger Restriction**

A trigger restriction specifies a Boolean (logical) expression that must be TRUE for the trigger to fire. The trigger action is not executed if the trigger restriction evaluates to FALSE or UNKNOWN.

A trigger restriction is an option available for triggers that are fired for each row. Its function is to control the execution of a trigger conditionally. You specify a trigger restriction using a WHEN clause. For example, the REORDER trigger in Figure 15 - 3 has a trigger restriction. The trigger is fired by an UPDATE statement affecting the PARTS\_ON\_HAND column of the INVENTORY table, but the trigger action only fires if the following expression is TRUE:

```
new.parts_on_hand < new.reorder_point
```

### **Trigger Action**

A trigger action is the procedure (PL/SQL block) that contains the SQL statements and PL/SQL code to be executed when a triggering statement is issued and the trigger restriction evaluates to TRUE.

Similar to stored procedures, a trigger action can contain SQL and PL/SQL statements, define PL/SQL language constructs (variables, constants, cursors, exceptions, and so on), and call stored procedures. Additionally, for row trigger, the statements in a trigger action have access to column values (new and old) of the current row being processed by the trigger. Two correlation names provide access to the old and new values for each column.

### **Types of Triggers**

When you define a trigger, you can specify the number of times the trigger action is to be executed: once for every row affected by the triggering statement (such as might be fired by an UPDATE statement that updates many rows), or once for the triggering statement, no matter how many rows it affects.

**Row Triggers** A row trigger is fired each time the table is affected by the triggering statement. For example, if an UPDATE statement updates multiple rows of a table, a row trigger is fired once for each row affected by the UPDATE statement. If a triggering statement affects no rows, a row trigger is not executed at all.

Row triggers are useful if the code in the trigger action depends on data provided by the triggering statement or rows that are affected. For example, Figure 15 - 3 illustrates a row trigger that uses the values of each row affected by the triggering statement.

**Statement Triggers** A statement trigger is fired once on behalf of the triggering statement, regardless of the number of rows in the table that the triggering statement affects (even if no rows are affected). For example, if a DELETE statement deletes several rows from a table, a statement-level DELETE trigger is fired only once, regardless of how many rows are deleted from the table.

Statement triggers are useful if the code in the trigger action does not depend on the data provided by the triggering statement or the rows affected. For example, if a trigger makes a complex security check on the current time or user, or if a trigger generates a single audit record based on the type of triggering statement, a statement trigger is used.

### **BEFORE vs. AFTER Triggers**

When defining a trigger, you can specify the trigger timing. That is, you can specify whether the trigger action is to be executed before or after the triggering statement. BEFORE and AFTER apply to both statement and row triggers.

BEFORE Triggers BEFORE triggers execute the trigger action before the triggering statement. This type of trigger is commonly used in the following situations:

- BEFORE triggers are used when the trigger action should determine whether the triggering statement should be allowed to complete. By using a BEFORE trigger for this purpose, you can eliminate unnecessary processing of the triggering statement and its eventual rollback in cases where an exception is raised in the trigger action.
- BEFORE triggers are used to derive specific column values before completing a triggering INSERT or UPDATE statement.

AFTER Triggers AFTER triggers execute the trigger action after the triggering statement is executed. AFTER triggers are used in the following situations:

- AFTER triggers are used when you want the triggering statement to complete before executing the trigger action.
- If a BEFORE trigger is already present, an AFTER trigger can perform different actions on the same triggering statement.

### **Combinations**

Using the options listed in the previous two sections, you can create four types of triggers:

- BEFORE statement trigger Before executing the triggering statement, the trigger action is executed.
- BEFORE row trigger Before modifying each row affected by the triggering statement and before checking appropriate integrity constraints, the trigger action is executed provided that the trigger restriction was not violated.
- AFTER statement trigger After executing the triggering statement and applying any deferred integrity constraints, the trigger action is executed.
- AFTER row trigger After modifying each row affected by the triggering statement and possibly applying appropriate integrity constraints, the trigger action is executed for the current row provided the trigger restriction was not violated. Unlike BEFORE row triggers, AFTER row triggers lock rows.

You can have multiple triggers of the same type for the same statement for any given table. For example you may have two BEFORE STATEMENT triggers for UPDATE statements on the EMP table. Multiple triggers of the same type permit modular installation of applications that have triggers on the same tables. Also, Oracle snapshot logs use AFTER ROW triggers, so you can design your own AFTER ROW trigger in addition to the Oracle-defined AFTER ROW trigger.

You can create as many triggers of the preceding different types as you need for each type of DML

statement (INSERT, UPDATE, or DELETE). For example, suppose you have a table, SAL, and you want to know when the table is being accessed and the types of queries being issued. A global session variable, STAT.ROWCNT, is initialized to zero by a BEFORE statement trigger, then it is increased each time the row trigger is executed, and finally the statistical information is saved in the table STAT\_TAB by the AFTER statement trigger.

**Example - ROW LEVEL AFTER UPDATE TRIGGER**

```
CREATE OR REPLACE TRIGGER "TRIGGER_ROW_LVL_AFTER_UPDATE"
AFTER UPDATE OF BOOK_STATUS_ID ON LIBRARY
FOR EACH ROW
BEGIN
INSERT INTO LIBRARY_AUDIT_ROW_LVL(BOOK_ID, BOOK_STATUS_ID,
UPDATE_TS) VALUES (:old.BOOK_ID, :old.BOOK_STATUS_ID,
CURRENT_TIMESTAMP);
END;
```

**Example - ROW LEVEL BEFORE UPDATE TRIGGER**

```
CREATE OR REPLACE TRIGGER "TRIGGER_ROW_LVL_BEFORE_UPDATE"
BEFORE UPDATE OF BOOK_STATUS_ID ON LIBRARY
FOR EACH ROW
BEGIN
INSERT INTO LIBRARY_AUDIT_ROW_LVL(BOOK_ID, BOOK_STATUS_ID,
UPDATE_TS) VALUES (:old.BOOK_ID, :old.BOOK_STATUS_ID,
CURRENT_TIMESTAMP);
END;
```

**Example - STATEMENT LEVEL AFTER DELETE TRIGGER**

```
CREATE OR REPLACE TRIGGER TRIGGER_STMT_LVL_AFTER_DELETE
AFTER DELETE ON LIBRARY
BEGIN
INSERT INTO LIBRARY_AUDIT_STMT_LVL (STMT_TYPE, UPDATE_TS) VALUES
('AFTER DELETE', CURRENT_TIMESTAMP);
END;
```

**Example - STATEMENT LEVEL BEFORE DELETE TRIGGER**

```
CREATE OR REPLACE TRIGGER TRIGGER_STMT_LVL_BEFORE_DELETE
BEFORE DELETE ON LIBRARY
BEGIN
INSERT INTO LIBRARY_AUDIT_STMT_LVL (STMT_TYPE, UPDATE_TS) VALUES
('BEFORE DELETE', CURRENT_TIMESTAMP);
END;
```

**Conclusion:** Here we understood what are triggers and events, types of triggers how triggers are used on database, and come to know that how use-full triggers are to manage the database.

## **Group B Large Scale Databases**

### **EXPERIMENT NO. B -1**

#### **Study of Open Source NOSQL Database**

**Title:-** Study of Open Source NOSQL Database: MongoDB (Installation, Basic CRUD operations, Execution)

**Objective:** Understand installation procedure of MongoDB and execution of basic CRUD operations.

**Prerequisites:**

Students should have a basic understanding of database, text editor and execution of programs, etc. Because we are going to develop high performance database, so it will be good if you have an understanding on the basic concepts of Database (RDBMS).

**Theory:** MongoDB is an open-source document database and leading NoSQL database. MongoDB is written in C++. This tutorial will give you great understanding on MongoDB concepts needed to create and deploy a highly scalable and performance-oriented database.

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 <code>_id</code> provided by mongodb itself)

## Installation Procedure in FEDORA 19

### Step #1: Add the MongoDB Repository

For a refresher on editing files with vim see: `vim /etc/yum.repos.d/mongodb.repo`

**Option A:** If you are running a 64-bit system, add the following information to the file you've created, using `i` to insert:

```
[mongodb]
name=MongoDBRepository
baseurl=http://downloads-distro.mongodb.org/repo/redhat/os/x86_64/
gpgcheck=0
enabled=1
```

Then exit and save the file with the command `:wq`.

**Option B:** If you are running a 32-bit system, add the following information to the file you've created, using `i` to insert:

```
[mongodb]
name=MongoDBRepository
baseurl=http://downloads-distro.mongodb.org/repo/redhat/os/i686/
gpgcheck=0
enabled=1
```

Then exit and save the file with the command `:wq`.

### Step #2: Install MongoDB

As a matter of best practice we'll update our packages:

```
yum -y update
```

At this point, installing MongoDB is as simple as running just one command:

```
yum -y install mongodb-org mongodb-org-server
```

### Step #3: Get MongoDB Running

Start-Up MongoDB

```
systemctl start mongod
```

Check MongoDB Service Status

```
systemctl status mongod
```

Start the MongoDB Service at Boot

```
systemctl enable mongod
```

### Summary List of Status Statistics (Continuous)

```
mongostat
```

### Summary List of Status Statistics (5 Rows, Summarized Every 2 Seconds)

```
mongostat --rowcount 5 2
```

### Enter the MongoDB Command Line

```
mongo
```

By default, running this command will look for a MongoDB server listening on port 27017 on the localhost interface. If you'd like to connect to a MongoDB server running on a different port, then use the `--port` option. For example, if you wanted to connect to a local MongoDB server listening on port 22222, then you'd issue the following command:

```
mongo --port 22222
```

### Shutdown MongoDB

```
systemctl stop mongod
```

## 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.

**Conclusion:** The installation procedure of MongoDB is successfully understood.

## EXPERIMENT NO. B -2

### CRUD Operations on MONGODB

**Title:** - Design and Develop MongoDB Queries using CRUD operations. (Use CRUD operations, SAVE method, logical operators)

**Objective:** Understand CRUD operations in MongoDB and execution of basic CRUD operations.

**Prerequisites:**

Students should have a basic understanding of database, text editor and execution of programs, etc. Because we are going to develop high performance database, so it will be good if you have an understanding on the basic concepts of Database (RDBMS) and Basics of mongodb.

**Theory:** CRUD operations *create, read, update, and delete* documents.

#### I. Create Operations

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

MongoDB provides the following methods to insert documents into a collection:

- `db.collection.insertOne()` *New in version 3.2*
- `db.collection.insertMany()` *New in version 3.2*

In MongoDB, insert operations target a single collection. All write operations in MongoDB are atomic on the level of a single document.

For examples, see [Insert Documents](#).

```
db.users.insertOne(  ← collection
{
  name: "sue",        ← field: value
  age: 26,            ← field: value
  status: "pending"   ← field: value
})                    } document
```

```
db.inventory.insertMany([
  { item: "journal", qty: 25, size: { h: 14, w: 21, uom: "cm" }, status: "A" },
  { item: "notebook", qty: 50, size: { h: 8.5, w: 11, uom: "in" }, status: "A" },
  { item: "paper", qty: 100, size: { h: 8.5, w: 11, uom: "in" }, status: "D" },
  { item: "planner", qty: 75, size: { h: 22.85, w: 30, uom: "cm" }, status: "D" },
  { item: "postcard", qty: 45, size: { h: 10, w: 15.25, uom: "cm" }, status: "A" }
]);
```

#### II. Read Operations

Read operations retrieve documents from a collection; i.e. queries a collection for documents. MongoDB provides the following methods to read documents from a collection:

- `db.collection.find()`

```
db.users.find(
  { age: { $gt: 18 } },
  { name: 1, address: 1 }
).limit(5)
```

← collection  
 ← query criteria  
 ← projection  
 ← cursor modifier

You can specify [query filters or criteria](#) that identify the documents to return.

### III. Update Operations

Update operations modify existing [documents](#) in a [collection](#).

For examples, see [Update Documents](#).

```
>db.TE.update({Roll:2},{ $set:{Name:"Wagholi" }});
```

Where 'TE' is Collection

MongoDB provides the following methods to update documents of a collection:

- [db.collection.updateOne\(\)](#) *New in version 3.2*
- [db.collection.updateMany\(\)](#) *New in version 3.2*
- [db.collection.replaceOne\(\)](#) *New in version 3.2*

In MongoDB, update operations target a single collection. All write operations in MongoDB are [atomic](#) on the level of a single document.

You can specify criteria, or filters, that identify the documents to update. These [filters](#) use the same syntax as read operations.

### IV. Delete Operations

Delete operations remove documents from a collection.

For examples:

```
> db.TE.remove({Roll:3});
```

MongoDB provides the following methods to delete documents of a collection:

- [db.collection.deleteOne\(\)](#) *New in version 3.2*
  - [db.collection.deleteMany\(\)](#) *New in version 3.2*
- In MongoDB, delete operations target a single [collection](#).

All write operations in MongoDB are [atomic](#) on the level of a single document.

You can specify criteria, or filters, that identify the documents to remove. These [filters](#) use the same syntax as read operations.

#### SAVE Method:

```
db.collection.save();
```

Updates an existing [document](#) or inserts a new document, depending on its [document](#) parameter.

The save() method has the following form:

The save() method uses either the [insert](#) or the [update](#) command, which use the default [write concern](#). To specify a different write concern, include the write concern in the options parameter.

#### Insert:

If the document does **not** contain an [\\_id](#) field, then the save() method calls the [insert\(\)](#) method. During the operation, the [mongo](#) shell will create an [ObjectId](#) and assign it to the [\\_id](#) field.

#### Update:

If the document contains an [\\_id](#) field, then the save() method is equivalent to an update with the [upsert option](#) set to **true** and the query predicate on the [\\_id](#) field.

## Examples

In the following example, `save()` method performs an insert since the document passed to the method does not contain the `_id` field:

```
> db.products.save( { item: "book", qty: 40 } );
```

In the following example, `save()` performs an update with `upsert:true` since the document contains an `_id` field:

```
> db.products.save( { _id: 100, item: "water", qty: 30 } );
```

# Logical Query Operators

Name	Description
<code>\$and</code>	Joins query clauses with a logical <b>AND</b> returns all documents that match the conditions of both clauses.
<code>\$not</code>	Inverts the effect of a query expression and returns documents that do <i>not</i> match the query expression
<code>\$nor</code>	Joins query clauses with a logical <b>NOR</b> returns all documents that fail to match both clauses.
<code>\$or</code>	Joins query clauses with a logical <b>OR</b> returns all documents that match the conditions of either clause.

## Example:

```
[student@localhost ~]$ su
Password:
[root@localhost student]# systemctl start mongod.service
[root@localhost student]# mongo
MongoDB shell version: 2.2.4
connecting to: test
> show dbs;
Bank    0.203125GB
local   (empty)
> use Bank;
switched to db Bank
> show tables;
Employee
system.indexes
> db.Employee.find();
{ "_id" : ObjectId("59688d2a015fef80611e763"), "R_no" : 1, "Name" : "ABC" }
{ "_id" : ObjectId("59688a32a015fef80611e764"), "R_no" : 2, "Name" : "pqr" }
> db.Employee.find({},{_id:0});
{ "R_no" : 1, "Name" : "ABC" }
```

```

{ "R_no" : 2, "Name" : "pqr" }
> db.Employee.find({R_no:1},{_id:0});
{ "R_no" : 1, "Name" : "ABC" }
> db.Employee.insert({R_no:3,Name:'XYZ'});
> db.Employee.insert({R_no:4,Name:'lmn'});
> db.Employee.find({}, {_id:0});
{ "R_no" : 1, "Name" : "ABC" }
{ "R_no" : 2, "Name" : "pqr" }
{ "R_no" : 3, "Name" : "XYZ" }
{ "R_no" : 4, "Name" : "lmn" }

> db.Employee.find({R_no:1});
{ "_id" : ObjectId("596888d2a015fef80611e763"), "R_no" : 1, "Name" : "ABC" }

> db.Employee.find({R_no:{ $in:[1,2]}});
{ "_id" : ObjectId("596888d2a015fef80611e763"), "R_no" : 1, "Name" : "ABC" }
{ "_id" : ObjectId("59688a32a015fef80611e764"), "R_no" : 2, "Name" : "pqr" }
> db.Employee.find({R_no:{ $not: { $in:[1,2]}} });
{ "_id" : ObjectId("59780a076510413ed9577b03"), "R_no" : 3, "Name" : "XYZ" }
{ "_id" : ObjectId("59780a106510413ed9577b04"), "R_no" : 4, "Name" : "lmn" }
> db.Employee.find( { $or :[{R_no:{ $gte:3}}, {Name:'lmn'}] });
{ "_id" : ObjectId("59780a076510413ed9577b03"), "R_no" : 3, "Name" : "XYZ" }
{ "_id" : ObjectId("59780a106510413ed9577b04"), "R_no" : 4, "Name" : "lmn" }
> db.Employee.find( { $or :[{R_no:{ $lte:3}}, {Name:'lmn'}] });
{ "_id" : ObjectId("596888d2a015fef80611e763"), "R_no" : 1, "Name" : "ABC" }
{ "_id" : ObjectId("59688a32a015fef80611e764"), "R_no" : 2, "Name" : "pqr" }
{ "_id" : ObjectId("59780a076510413ed9577b03"), "R_no" : 3, "Name" : "XYZ" }
{ "_id" : ObjectId("59780a106510413ed9577b04"), "R_no" : 4, "Name" : "lmn" }
> db.Employee.find( { $and :[{R_no:4}, {Name:'lmn'}] });
{ "_id" : ObjectId("59780a106510413ed9577b04"), "R_no" : 4, "Name" : "lmn" }

```

**Conclusion:** The CRUD Operations SAVE Method and Logical operators in MongoDB is successfully understood.

### Sample Command for Basic Operations:

```
[Vilas@localhost bin]$ ./mongo
```

```
MongoDB shell version: 2.6.1
```

```
connecting to: test
```

```
> show databases;
```

```
VBK      0.078GB
```

```
admin    (empty)
```

```
local    0.078GB
```

```
newsletter 0.078GB
```

```
> use COEM // Created new Database named as "COEM"
```

```
switched to db COEM
```

```
> db.createCollection("TE") // Created new Collection named as "TE"
```

```
{ "ok" : 1 }
```

```
> db.TE.insert({Roll:1,Name:"ABC",Address:"Pune",Per:76})
```

```
WriteResult({ "nInserted" : 1 })
```

```
> db.TE.insert({Roll:2,Name:"PQR",Address:"Pune",Per:75})
```

```
WriteResult({ "nInserted" : 1 })
```

```
> db.TE.insert({Roll:3,Name:"LMN",Address:"Hadapsar",Per:70})
```

```
WriteResult({ "nInserted" : 1 })
```

```
> db.TE.find({})
```

```
{ "_id" : ObjectId("541963be2741c7552caef0a9"), "Roll" : 1, "Name" : "ABC", "Address" : "Pune",  
"Per" : 76 }
```

```
{ "_id" : ObjectId("541963cb2741c7552caef0aa"), "Roll" : 2, "Name" : "PQR", "Address" : "Pune",  
"Per" : 75 }
```

```
{ "_id" : ObjectId("541963dc2741c7552caef0ab"), "Roll" : 3, "Name" : "LMN", "Address" :  
"Hadapsar", "Per" : 70 }
```

```
> db.TE.find({})
```

```
{ "_id" : ObjectId("541963be2741c7552caef0a9"), "Roll" : 1, "Name" : "ABC", "Address" : "Pune",  
"Per" : 76 }
```

```
{ "_id" : ObjectId("541963cb2741c7552caef0aa"), "Roll" : 2, "Name" : "PQR", "Address" : "Pune",  
"Per" : 75 }
```

```
{ "_id" : ObjectId("541963dc2741c7552caef0ab"), "Roll" : 3, "Name" : "LMN", "Address" :  
"Hadapsar", "Per" : 70 }
```

```
> db.TE.update({Roll:2},{ $set:{Name:"Wagholi" } })
```

```
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

```
> db.TE.find({})
```

```
{ "_id" : ObjectId("541963be2741c7552caef0a9"), "Roll" : 1, "Name" : "ABC", "Address" : "Pune",  
"Per" : 76 }  
{ "_id" : ObjectId("541963cb2741c7552caef0aa"), "Roll" : 2, "Name" : "Wagholi", "Address" :  
"Pune", "Per" : 75 }  
{ "_id" : ObjectId("541963dc2741c7552caef0ab"), "Roll" : 3, "Name" : "LMN", "Address" :  
"Hadapsar", "Per" : 70 }
```

```
> db.TE.remove({Roll:3})  
WriteResult({ "nRemoved" : 1 })
```

```
> db.TE.find({})  
{ "_id" : ObjectId("541963be2741c7552caef0a9"), "Roll" : 1, "Name" : "ABC", "Address" : "Pune",  
"Per" : 76 }  
{ "_id" : ObjectId("541963cb2741c7552caef0aa"), "Roll" : 2, "Name" : "Wagholi", "Address" :  
"Pune", "Per" : 75 }
```

```
> db.TE.drop()  
true
```

```
> db.TE.find({})
```

```
> show collections  
system.indexes
```

```
> show databases;  
COEM      0.078GB  
VBK       0.078GB  
admin     (empty)  
local     0.078GB  
newsletter 0.078GB
```

```
> db.dropDatabase("COEM")  
2014-09-17T16:17:18.278+0530 dropDatabase doesn't take arguments at src/mongo/shell/db.js:141
```

```
> db.dropDatabase()  
{ "dropped" : "COEM", "ok" : 1 }
```

```
> show databases;  
VBK       0.078GB  
admin     (empty)  
local     0.078GB  
newsletter 0.078GB
```



## EXPERIMENT NO. B -3

### Aggregation and indexing in MONGODB

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

**Objective:** Understand aggregation and indexing operations in MongoDB

**Prerequisites:**

Basics of mongodb.

**Theory:** Aggregation 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. MongoDB provides three ways to perform aggregation: the [aggregation pipeline](#), the [map-reduce function](#), and [single purpose aggregation methods](#).

MongoDB's aggregation framework is modeled on the concept of data processing pipelines. Documents enter a multi-stage pipeline that transforms the documents into an aggregated result. The most basic pipeline stages provide filters that operate like queries and document transformations that modify the form of the output document.

Other pipeline operations provide tools for grouping and sorting documents by specific field or fields as well as tools for aggregating the contents of arrays, including arrays of documents. In addition, pipeline stages can use operators for tasks such as calculating the average or concatenating a string. The pipeline provides efficient data aggregation using native operations within MongoDB, and is the preferred method for data aggregation in MongoDB.

The aggregation pipeline can operate on a sharded collection. The aggregation pipeline can use indexes to improve its performance during some of its stages. In addition, the aggregation pipeline has an internal optimization phase. See Pipeline Operators and Indexes and Aggregation Pipeline Optimization for details.

```
Collection
  ↓
db.orders.aggregate( [
  $match stage → { $match: { status: "A" } },
  $group stage → { $group: { _id: "$cust_id", total: { $sum: "$amount" } } }
] )
```

Examples:

```
> db.student.find({})
{ "_id" : ObjectId("53fac0cd3343ed16abd76169"), "Rno" : 1, "Name" : "A", "Dept" : "Civil", "Marks" : 55 }
{ "_id" : ObjectId("53fac0da3343ed16abd7616a"), "Rno" : 2, "Name" : "B", "Dept" : "Civil", "Marks" : 56 }
{ "_id" : ObjectId("53fac0ed3343ed16abd7616b"), "Rno" : 3, "Name" : "C", "Dept" : "Civil", "Marks" : 60 }
{ "_id" : ObjectId("53fac1013343ed16abd7616c"), "Rno" : 4, "Name" : "P", "Dept" : "Comp", "Marks" : 60 }
{ "_id" : ObjectId("53fac10e3343ed16abd7616d"), "Rno" : 5, "Name" : "Q", "Dept" : "Comp", "Marks" : 61 }
{ "_id" : ObjectId("53fac11a3343ed16abd7616e"), "Rno" : 6, "Name" : "R", "Dept" : "Comp", "Marks" : 61 }
{ "_id" : ObjectId("53fac12e3343ed16abd7616f"), "Rno" : 7, "Name" : "S", "Dept" : "Comp", "Marks" : 62 }
```

```

{ "_id" : ObjectId("53fac1433343ed16abd76170"), "Rno" : 8, "Name" : "L", "Dept" : "Mech", "Marks" : 52 }
{ "_id" : ObjectId("53fac14c3343ed16abd76171"), "Rno" : 9, "Name" : "M", "Dept" : "Mech", "Marks" : 52 }
{ "_id" : ObjectId("53fac18f3343ed16abd76172"), "Rno" : 10, "Name" : "T", "Dept" : "Comp", "Marks" : 53 }
}
{ "_id" : ObjectId("53fac1a33343ed16abd76173"), "Rno" : 11, "Name" : "D", "Dept" : "Civil", "Marks" : 65 }
> db.student.find({})
{ "_id" : ObjectId("53fac0cd3343ed16abd76169"), "Rno" : 1, "Name" : "A", "Dept" : "Civil", "Marks" : 55 }
{ "_id" : ObjectId("53fac0da3343ed16abd7616a"), "Rno" : 2, "Name" : "B", "Dept" : "Civil", "Marks" : 56 }
{ "_id" : ObjectId("53fac0ed3343ed16abd7616b"), "Rno" : 3, "Name" : "C", "Dept" : "Civil", "Marks" : 60 }
{ "_id" : ObjectId("53fac1013343ed16abd7616c"), "Rno" : 4, "Name" : "P", "Dept" : "Comp", "Marks" : 60 }
{ "_id" : ObjectId("53fac10e3343ed16abd7616d"), "Rno" : 5, "Name" : "Q", "Dept" : "Comp", "Marks" : 61 }
{ "_id" : ObjectId("53fac11a3343ed16abd7616e"), "Rno" : 6, "Name" : "R", "Dept" : "Comp", "Marks" : 61 }
{ "_id" : ObjectId("53fac12e3343ed16abd7616f"), "Rno" : 7, "Name" : "S", "Dept" : "Comp", "Marks" : 62 }
{ "_id" : ObjectId("53fac1433343ed16abd76170"), "Rno" : 8, "Name" : "L", "Dept" : "Mech", "Marks" : 52 }
{ "_id" : ObjectId("53fac14c3343ed16abd76171"), "Rno" : 9, "Name" : "M", "Dept" : "Mech", "Marks" : 52 }
{ "_id" : ObjectId("53fac18f3343ed16abd76172"), "Rno" : 10, "Name" : "T", "Dept" : "Comp", "Marks" : 53 }
}
{ "_id" : ObjectId("53fac1a33343ed16abd76173"), "Rno" : 11, "Name" : "D", "Dept" : "Civil", "Marks" : 65 }
> db.student.aggregate({ $project: { Rno:1 } })
{ "_id" : ObjectId("53fac0cd3343ed16abd76169"), "Rno" : 1 }
{ "_id" : ObjectId("53fac0da3343ed16abd7616a"), "Rno" : 2 }
{ "_id" : ObjectId("53fac0ed3343ed16abd7616b"), "Rno" : 3 }
{ "_id" : ObjectId("53fac1013343ed16abd7616c"), "Rno" : 4 }
{ "_id" : ObjectId("53fac10e3343ed16abd7616d"), "Rno" : 5 }
{ "_id" : ObjectId("53fac11a3343ed16abd7616e"), "Rno" : 6 }
{ "_id" : ObjectId("53fac12e3343ed16abd7616f"), "Rno" : 7 }
{ "_id" : ObjectId("53fac1433343ed16abd76170"), "Rno" : 8 }
{ "_id" : ObjectId("53fac14c3343ed16abd76171"), "Rno" : 9 }
{ "_id" : ObjectId("53fac18f3343ed16abd76172"), "Rno" : 10 }
{ "_id" : ObjectId("53fac1a33343ed16abd76173"), "Rno" : 11 }
> db.student.aggregate({ $project: { Name:1 } })
{ "_id" : ObjectId("53fac0cd3343ed16abd76169"), "Name" : "A" }
{ "_id" : ObjectId("53fac0da3343ed16abd7616a"), "Name" : "B" }
{ "_id" : ObjectId("53fac0ed3343ed16abd7616b"), "Name" : "C" }
{ "_id" : ObjectId("53fac1013343ed16abd7616c"), "Name" : "P" }
{ "_id" : ObjectId("53fac10e3343ed16abd7616d"), "Name" : "Q" }
{ "_id" : ObjectId("53fac11a3343ed16abd7616e"), "Name" : "R" }
{ "_id" : ObjectId("53fac12e3343ed16abd7616f"), "Name" : "S" }
{ "_id" : ObjectId("53fac1433343ed16abd76170"), "Name" : "L" }
{ "_id" : ObjectId("53fac14c3343ed16abd76171"), "Name" : "M" }
{ "_id" : ObjectId("53fac18f3343ed16abd76172"), "Name" : "T" }
{ "_id" : ObjectId("53fac1a33343ed16abd76173"), "Name" : "D" }
> db.student.aggregate({ $sort: { Name:1 } })
{ "_id" : ObjectId("53fac0cd3343ed16abd76169"), "Rno" : 1, "Name" : "A", "Dept" : "Civil", "Marks" : 55 }
{ "_id" : ObjectId("53fac0da3343ed16abd7616a"), "Rno" : 2, "Name" : "B", "Dept" : "Civil", "Marks" : 56 }
{ "_id" : ObjectId("53fac0ed3343ed16abd7616b"), "Rno" : 3, "Name" : "C", "Dept" : "Civil", "Marks" : 60 }
{ "_id" : ObjectId("53fac1a33343ed16abd76173"), "Rno" : 11, "Name" : "D", "Dept" : "Civil", "Marks" : 65 }
{ "_id" : ObjectId("53fac1433343ed16abd76170"), "Rno" : 8, "Name" : "L", "Dept" : "Mech", "Marks" : 52 }
{ "_id" : ObjectId("53fac14c3343ed16abd76171"), "Rno" : 9, "Name" : "M", "Dept" : "Mech", "Marks" : 52 }
{ "_id" : ObjectId("53fac1013343ed16abd7616c"), "Rno" : 4, "Name" : "P", "Dept" : "Comp", "Marks" : 60 }
{ "_id" : ObjectId("53fac10e3343ed16abd7616d"), "Rno" : 5, "Name" : "Q", "Dept" : "Comp", "Marks" : 61 }
{ "_id" : ObjectId("53fac11a3343ed16abd7616e"), "Rno" : 6, "Name" : "R", "Dept" : "Comp", "Marks" : 61 }
{ "_id" : ObjectId("53fac12e3343ed16abd7616f"), "Rno" : 7, "Name" : "S", "Dept" : "Comp", "Marks" : 62 }
{ "_id" : ObjectId("53fac18f3343ed16abd76172"), "Rno" : 10, "Name" : "T", "Dept" : "Comp", "Marks" : 53 }
}

```

[illegible]

```

> db.student.aggregate({$sort:{Marks:-1}},{$limit:4})
{ "_id" : ObjectId("53fac1a3343ed16abd76173"), "Rno" : 11, "Name" : "D", "Dept" : "Civil", "Marks" : 65 }
{ "_id" : ObjectId("53fac12e3343ed16abd7616f"), "Rno" : 7, "Name" : "S", "Dept" : "Comp", "Marks" : 62 }
{ "_id" : ObjectId("53fac11a3343ed16abd7616e"), "Rno" : 6, "Name" : "R", "Dept" : "Comp", "Marks" : 61 }
{ "_id" : ObjectId("53fac10e3343ed16abd7616d"), "Rno" : 5, "Name" : "Q", "Dept" : "Comp", "Marks" : 61 }

> db.student.aggregate({$group:{"_id":"$Dept","Count":{$sum:1}}})
{ "_id" : "Mech", "Count" : 2 }
{ "_id" : "Comp", "Count" : 5 }
{ "_id" : "Civil", "Count" : 4 }

> db.student.aggregate({$group:{"_id":"$Marks","Count":{$sum:1}}})
{ "_id" : 65, "Count" : 1 }
{ "_id" : 52, "Count" : 2 }
{ "_id" : 61, "Count" : 2 }
{ "_id" : 53, "Count" : 1 }
{ "_id" : 60, "Count" : 2 }
{ "_id" : 56, "Count" : 1 }
{ "_id" : 62, "Count" : 1 }
{ "_id" : 55, "Count" : 1 }

> db.student.aggregate({$group:{"_id":"$Marks","Count":{$avg:1}}})
{ "_id" : 65, "Count" : 1 }
{ "_id" : 52, "Count" : 1 }
{ "_id" : 61, "Count" : 1 }
{ "_id" : 53, "Count" : 1 }
{ "_id" : 60, "Count" : 1 }
{ "_id" : 56, "Count" : 1 }
{ "_id" : 62, "Count" : 1 }
{ "_id" : 55, "Count" : 1 }

> db.student.find({Dept:"Civil"})
{ "_id" : ObjectId("53fac0cd3343ed16abd76169"), "Rno" : 1, "Name" : "A", "Dept" : "Civil", "Marks" : 55 }
{ "_id" : ObjectId("53fac0da3343ed16abd7616a"), "Rno" : 2, "Name" : "B", "Dept" : "Civil", "Marks" : 56 }
{ "_id" : ObjectId("53fac0ed3343ed16abd7616b"), "Rno" : 3, "Name" : "C", "Dept" : "Civil", "Marks" : 60 }
{ "_id" : ObjectId("53fac1a3343ed16abd76173"), "Rno" : 11, "Name" : "D", "Dept" : "Civil", "Marks" : 65 }

> db.student.find({Dept:"Comp"})
{ "_id" : ObjectId("53fac1013343ed16abd7616c"), "Rno" : 4, "Name" : "P", "Dept" : "Comp", "Marks" : 60 }
{ "_id" : ObjectId("53fac10e3343ed16abd7616d"), "Rno" : 5, "Name" : "Q", "Dept" : "Comp", "Marks" : 61 }
{ "_id" : ObjectId("53fac11a3343ed16abd7616e"), "Rno" : 6, "Name" : "R", "Dept" : "Comp", "Marks" : 61 }
{ "_id" : ObjectId("53fac12e3343ed16abd7616f"), "Rno" : 7, "Name" : "S", "Dept" : "Comp", "Marks" : 62 }
{ "_id" : ObjectId("53fac18f3343ed16abd76172"), "Rno" : 10, "Name" : "T", "Dept" : "Comp", "Marks" : 53 }

> db.student.find({$or:[{Dept:"Comp"},{Dept:"Civil"}]})
{ "_id" : ObjectId("53fac0cd3343ed16abd76169"), "Rno" : 1, "Name" : "A", "Dept" : "Civil", "Marks" : 55 }
{ "_id" : ObjectId("53fac0da3343ed16abd7616a"), "Rno" : 2, "Name" : "B", "Dept" : "Civil", "Marks" : 56 }
{ "_id" : ObjectId("53fac0ed3343ed16abd7616b"), "Rno" : 3, "Name" : "C", "Dept" : "Civil", "Marks" : 60 }
{ "_id" : ObjectId("53fac1013343ed16abd7616c"), "Rno" : 4, "Name" : "P", "Dept" : "Comp", "Marks" : 60 }
{ "_id" : ObjectId("53fac10e3343ed16abd7616d"), "Rno" : 5, "Name" : "Q", "Dept" : "Comp", "Marks" : 61 }
{ "_id" : ObjectId("53fac11a3343ed16abd7616e"), "Rno" : 6, "Name" : "R", "Dept" : "Comp", "Marks" : 61 }
{ "_id" : ObjectId("53fac12e3343ed16abd7616f"), "Rno" : 7, "Name" : "S", "Dept" : "Comp", "Marks" : 62 }
{ "_id" : ObjectId("53fac18f3343ed16abd76172"), "Rno" : 10, "Name" : "T", "Dept" : "Comp", "Marks" : 53 }
{ "_id" : ObjectId("53fac1a3343ed16abd76173"), "Rno" : 11, "Name" : "D", "Dept" : "Civil", "Marks" : 65 }

> db.runCommand({aggregate:"student",pipeline:[{$project:{Rno:1}}]})
{

```

```

    "result" : [
      {
        "_id" : ObjectId("53fac0cd3343ed16abd76169"),
        "Rno" : 1
      },
      {
        "_id" : ObjectId("53fac0da3343ed16abd7616a"),
        "Rno" : 2
      },
      {
        "_id" : ObjectId("53fac0ed3343ed16abd7616b"),
        "Rno" : 3
      },
      {
        "_id" : ObjectId("53fac1013343ed16abd7616c"),
        "Rno" : 4
      },
      {
        "_id" : ObjectId("53fac10e3343ed16abd7616d"),
        "Rno" : 5
      },
      {
        "_id" : ObjectId("53fac11a3343ed16abd7616e"),
        "Rno" : 6
      },
      {
        "_id" : ObjectId("53fac12e3343ed16abd7616f"),
        "Rno" : 7
      },
      {
        "_id" : ObjectId("53fac1433343ed16abd76170"),
        "Rno" : 8
      },
      {
        "_id" : ObjectId("53fac14c3343ed16abd76171"),
        "Rno" : 9
      },
      {
        "_id" : ObjectId("53fac18f3343ed16abd76172"),
        "Rno" : 10
      },
      {
        "_id" : ObjectId("53fac1a33343ed16abd76173"),
        "Rno" : 11
      }
    ],
    "ok" : 1
  }
}
> db.runCommand({aggregate:"student",pipeline:[{$project:{Rno:1}},{$limit:3}}])
{
  "result" : [
    {
      "_id" : ObjectId("53fac0cd3343ed16abd76169"),
      "Rno" : 1
    },
  ],

```

```

        {
            "_id" : ObjectId("53fac0da3343ed16abd7616a"),
            "Rno" : 2
        },
        {
            "_id" : ObjectId("53fac0ed3343ed16abd7616b"),
            "Rno" : 3
        }
    ],
    "ok" : 1
}
> db.student.aggregate({$group:{"_id":"$Dept","AVERAGE":{$avg:"$Marks"}}})
{ "_id" : "Mech", "AVERAGE" : 52 }
{ "_id" : "Comp", "AVERAGE" : 59.4 }
{ "_id" : "Civil", "AVERAGE" : 59 }

> db.student.aggregate({$group:{"_id":"$Dept","AVERAGE":{$avg:"$Marks"}}})
{ "_id" : "Mech", "AVERAGE" : 52 }
{ "_id" : "Comp", "AVERAGE" : 59.4 }
{ "_id" : "Civil", "AVERAGE" : 59 }

> db.student.aggregate({$group:{"_id":"$Dept","MINIMUM":{$min:"$Marks"}}})
{ "_id" : "Mech", "MINIMUM" : 52 }
{ "_id" : "Comp", "MINIMUM" : 53 }
{ "_id" : "Civil", "MINIMUM" : 55 }
> db.student.aggregate({$group:{"_id":"$Dept","MAXIMUM":{$max:"$Marks"}}})
{ "_id" : "Mech", "MAXIMUM" : 52 }
{ "_id" : "Comp", "MAXIMUM" : 62 }
{ "_id" : "Civil", "MAXIMUM" : 65 }
> db.student.aggregate({$group:{"_id":"$Dept","TOTAL":{$sum:"$Marks"}}})
{ "_id" : "Mech", "TOTAL" : 104 }
{ "_id" : "Comp", "TOTAL" : 297 }
{ "_id" : "Civil", "TOTAL" : 236 }
> db.student.aggregate({$group:{"_id":"$Dept","COUNT":{$sum:1}}})
{ "_id" : "Mech", "COUNT" : 2 }
{ "_id" : "Comp", "COUNT" : 5 }
{ "_id" : "Civil", "COUNT" : 4 }
> db.student.aggregate({$group:{"_id":"$Dept","First":{$first:"$Marks"}}})
{ "_id" : "Mech", "First" : 52 }
{ "_id" : "Comp", "First" : 60 }
{ "_id" : "Civil", "First" : 55 }
> db.student.aggregate({$group:{"_id":"$Dept","Last":{$last:"$Marks"}}})
{ "_id" : "Mech", "Last" : 52 }
{ "_id" : "Comp", "Last" : 53 }
{ "_id" : "Civil", "Last" : 65 }
> db.student.aggregate({$limit:4},{ $sort:{Rno:1}})
{ "_id" : ObjectId("53fac0cd3343ed16abd76169"), "Rno" : 1, "Name" : "A", "Dept" : "Civil", "Marks" : 55 }
{ "_id" : ObjectId("53fac0da3343ed16abd7616a"), "Rno" : 2, "Name" : "B", "Dept" : "Civil", "Marks" : 56 }
{ "_id" : ObjectId("53fac0ed3343ed16abd7616b"), "Rno" : 3, "Name" : "C", "Dept" : "Civil", "Marks" : 60 }
{ "_id" : ObjectId("53fac1013343ed16abd7616c"), "Rno" : 4, "Name" : "P", "Dept" : "Comp", "Marks" : 60 }
> db.student.aggregate({$skip:2},{ $sort:{Rno:1}})
{ "_id" : ObjectId("53fac0ed3343ed16abd7616b"), "Rno" : 3, "Name" : "C", "Dept" : "Civil", "Marks" : 60 }
{ "_id" : ObjectId("53fac1013343ed16abd7616c"), "Rno" : 4, "Name" : "P", "Dept" : "Comp", "Marks" : 60 }
{ "_id" : ObjectId("53fac10e3343ed16abd7616d"), "Rno" : 5, "Name" : "Q", "Dept" : "Comp", "Marks" : 61 }
{ "_id" : ObjectId("53fac11a3343ed16abd7616e"), "Rno" : 6, "Name" : "R", "Dept" : "Comp", "Marks" : 61 }

```

```

{ "_id" : ObjectId("53fac12e3343ed16abd7616f"), "Rno" : 7, "Name" : "S", "Dept" : "Comp", "Marks" : 62 }
{ "_id" : ObjectId("53fac143343ed16abd76170"), "Rno" : 8, "Name" : "L", "Dept" : "Mech", "Marks" : 52 }
{ "_id" : ObjectId("53fac14c3343ed16abd76171"), "Rno" : 9, "Name" : "M", "Dept" : "Mech", "Marks" : 52 }
{ "_id" : ObjectId("53fac18f3343ed16abd76172"), "Rno" : 10, "Name" : "T", "Dept" : "Comp", "Marks" : 53 }
}
{ "_id" : ObjectId("53fac1a3343ed16abd76173"), "Rno" : 11, "Name" : "D", "Dept" : "Civil", "Marks" : 65 }
> db.student.aggregate({$skip:2},{ $sort:{Rno:1}})
{ "_id" : ObjectId("53fac0ed3343ed16abd7616b"), "Rno" : 3, "Name" : "C", "Dept" : "Civil", "Marks" : 60 }
{ "_id" : ObjectId("53fac1013343ed16abd7616c"), "Rno" : 4, "Name" : "P", "Dept" : "Comp", "Marks" : 60 }
{ "_id" : ObjectId("53fac10e3343ed16abd7616d"), "Rno" : 5, "Name" : "Q", "Dept" : "Comp", "Marks" : 61 }
{ "_id" : ObjectId("53fac11a3343ed16abd7616e"), "Rno" : 6, "Name" : "R", "Dept" : "Comp", "Marks" : 61 }
{ "_id" : ObjectId("53fac12e3343ed16abd7616f"), "Rno" : 7, "Name" : "S", "Dept" : "Comp", "Marks" : 62 }
{ "_id" : ObjectId("53fac143343ed16abd76170"), "Rno" : 8, "Name" : "L", "Dept" : "Mech", "Marks" : 52 }
{ "_id" : ObjectId("53fac14c3343ed16abd76171"), "Rno" : 9, "Name" : "M", "Dept" : "Mech", "Marks" : 52 }
{ "_id" : ObjectId("53fac18f3343ed16abd76172"), "Rno" : 10, "Name" : "T", "Dept" : "Comp", "Marks" : 53 }
}
{ "_id" : ObjectId("53fac1a3343ed16abd76173"), "Rno" : 11, "Name" : "D", "Dept" : "Civil", "Marks" : 65 }
> db.student.aggregate({ $sort:{Rno:1}},{ $skip:2})
{ "_id" : ObjectId("53fac0ed3343ed16abd7616b"), "Rno" : 3, "Name" : "C", "Dept" : "Civil", "Marks" : 60 }
{ "_id" : ObjectId("53fac1013343ed16abd7616c"), "Rno" : 4, "Name" : "P", "Dept" : "Comp", "Marks" : 60 }
{ "_id" : ObjectId("53fac10e3343ed16abd7616d"), "Rno" : 5, "Name" : "Q", "Dept" : "Comp", "Marks" : 61 }
{ "_id" : ObjectId("53fac11a3343ed16abd7616e"), "Rno" : 6, "Name" : "R", "Dept" : "Comp", "Marks" : 61 }
{ "_id" : ObjectId("53fac12e3343ed16abd7616f"), "Rno" : 7, "Name" : "S", "Dept" : "Comp", "Marks" : 62 }
{ "_id" : ObjectId("53fac143343ed16abd76170"), "Rno" : 8, "Name" : "L", "Dept" : "Mech", "Marks" : 52 }
{ "_id" : ObjectId("53fac14c3343ed16abd76171"), "Rno" : 9, "Name" : "M", "Dept" : "Mech", "Marks" : 52 }
{ "_id" : ObjectId("53fac18f3343ed16abd76172"), "Rno" : 10, "Name" : "T", "Dept" : "Comp", "Marks" : 53 }
}
{ "_id" : ObjectId("53fac1a3343ed16abd76173"), "Rno" : 11, "Name" : "D", "Dept" : "Civil", "Marks" : 65 }
> db.student.aggregate({ $sort:{Rno:-1}},{ $skip:2})
{ "_id" : ObjectId("53fac14c3343ed16abd76171"), "Rno" : 9, "Name" : "M", "Dept" : "Mech", "Marks" : 52 }
{ "_id" : ObjectId("53fac143343ed16abd76170"), "Rno" : 8, "Name" : "L", "Dept" : "Mech", "Marks" : 52 }
{ "_id" : ObjectId("53fac12e3343ed16abd7616f"), "Rno" : 7, "Name" : "S", "Dept" : "Comp", "Marks" : 62 }
{ "_id" : ObjectId("53fac11a3343ed16abd7616e"), "Rno" : 6, "Name" : "R", "Dept" : "Comp", "Marks" : 61 }
{ "_id" : ObjectId("53fac10e3343ed16abd7616d"), "Rno" : 5, "Name" : "Q", "Dept" : "Comp", "Marks" : 61 }
{ "_id" : ObjectId("53fac1013343ed16abd7616c"), "Rno" : 4, "Name" : "P", "Dept" : "Comp", "Marks" : 60 }
{ "_id" : ObjectId("53fac0ed3343ed16abd7616b"), "Rno" : 3, "Name" : "C", "Dept" : "Civil", "Marks" : 60 }
{ "_id" : ObjectId("53fac0da3343ed16abd7616a"), "Rno" : 2, "Name" : "B", "Dept" : "Civil", "Marks" : 56 }
{ "_id" : ObjectId("53fac0cd3343ed16abd76169"), "Rno" : 1, "Name" : "A", "Dept" : "Civil", "Marks" : 55 }

> db.tea.find({})
{ "_id" : ObjectId("53fabda13343ed16abd76165"), "rno" : 1, "name" : "vilas" }
{ "_id" : ObjectId("53fabded3343ed16abd76166"), "rno" : 2, "name" : "vilas", "dept" : "Comp" }
{ "_id" : ObjectId("53fabdfa3343ed16abd76167"), "rno" : 3, "name" : "PQR", "dept" : "Comp" }
{ "_id" : ObjectId("53fabe073343ed16abd76168"), "rno" : 4, "name" : "ABC", "dept" : "Civil" }

> db.Ass15.insert({Rno:1,Name:"ABC",Mobile:[99999,8888]})
WriteResult({ "nInserted" : 1 })
> db.Ass15.find({})
{ "_id" : ObjectId("53feb6b04ff96d10965820c2"), "Rno" : 1, "Name" : "ABC", "Mobile" : [ 99999, 8888 ] }
> db.Ass15.insert({Rno:2,Name:"PQR",Mobile:[22222,55555]})
WriteResult({ "nInserted" : 1 })
> db.Ass15.find({})
{ "_id" : ObjectId("53feb6b04ff96d10965820c2"), "Rno" : 1, "Name" : "ABC", "Mobile" : [ 99999, 8888 ] }
{ "_id" : ObjectId("53feb6da4ff96d10965820c3"), "Rno" : 2, "Name" : "PQR", "Mobile" : [ 22222, 55555 ] }
}

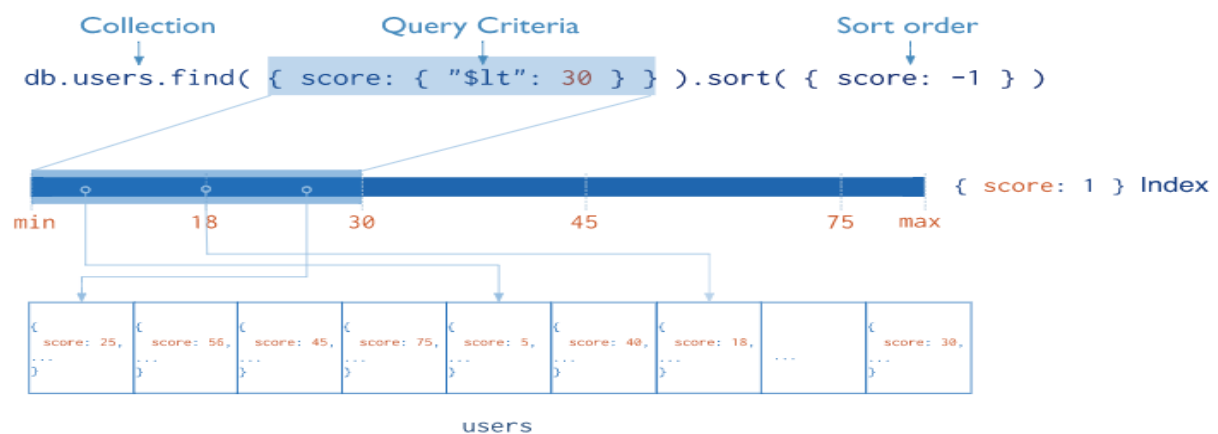
```

```
> db.Ass15.aggregate({$unwind:"$Mobile"})
{ "_id" : ObjectId("53feb6b04ff96d10965820c2"), "Rno" : 1, "Name" : "ABC", "Mobile" : 99999 }
{ "_id" : ObjectId("53feb6b04ff96d10965820c2"), "Rno" : 1, "Name" : "ABC", "Mobile" : 8888 }
{ "_id" : ObjectId("53feb6da4ff96d10965820c3"), "Rno" : 2, "Name" : "PQR", "Mobile" : 22222 }
{ "_id" : ObjectId("53feb6da4ff96d10965820c3"), "Rno" : 2, "Name" : "PQR", "Mobile" : 55555 }
```

## INDEXES:

Indexes support the efficient execution of queries in MongoDB. Without indexes, MongoDB must perform a *collection scan*, i.e. scan every document in a collection, to select those documents that match the query statement. If an appropriate index exists for a query, MongoDB can use the index to limit the number of documents it must inspect.

Indexes are special data structures [\[1\]](#) that store a small portion of the collection's data set in an easy to traverse form. The index stores the value of a specific field or set of fields, ordered by the value of the field. The ordering of the index entries supports efficient equality matches and range-based query operations. In addition, MongoDB can return sorted results by using the ordering in the index. The following diagram illustrates a query that selects and orders the matching documents using an index:



Fundamentally, indexes in MongoDB are similar to indexes in other database systems. MongoDB defines indexes at the [collection](#) level and supports indexes on any field or sub-field of the documents in a MongoDB collection.

### Default `_id` Index

MongoDB creates a [unique index](#) on the `_id` field during the creation of a collection. The `_id` index prevents clients from inserting two documents with the same value for the `_id` field. You cannot drop this index on the `_id` field.

### Create an Index

```
db.collection.createIndex( <key and index type specification>, <options> )
```

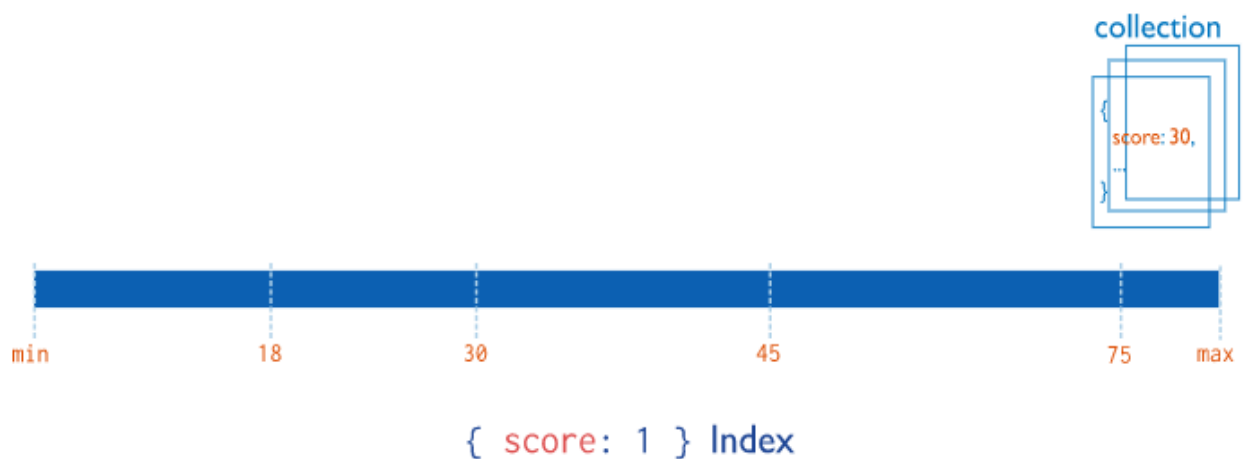
### Index Types



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

## Single Field

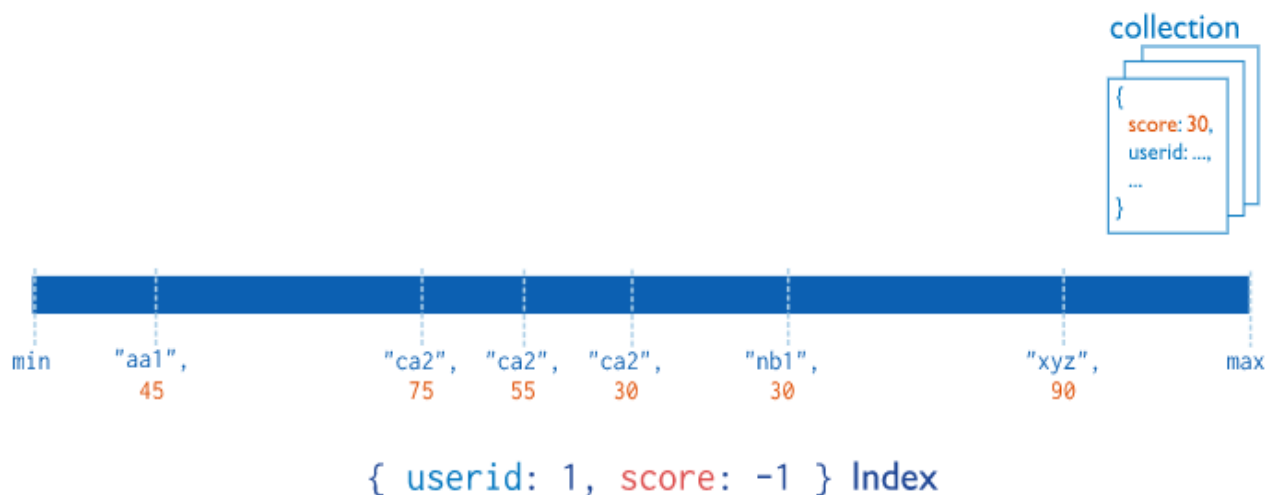
In addition to the MongoDB-defined `_id` index, MongoDB supports the creation of user-defined ascending/descending indexes on a [single field of a document](#).



## Compound Index

MongoDB also supports user-defined indexes on multiple fields, i.e. [compound indexes](#).

The order of fields listed in a compound index has significance. For instance, if a compound index consists of `{ userid: 1, score: -1 }`, the index sorts first by `userid` and then, within each `userid` value, sorts by `score`.



Examples:

```

root@Afroz:~#service mongod start
root@Afroz:~# mongo
MongoDB shell version: 3.2.11
connecting to: test
> show dbs
afroz  0.000GB
employee 0.000GB
local  0.000GB
> use employee
switched to db employee
> use employee
switched to db employee
> show tables;
doc
emp
> db.emp.find({}, {_id:0})
{ "empID" : 111, "name" : "pratik", "design" : "ceo", "salary" : 3000 }
{ "empID" : 112, "name" : "jatin", "design" : "manager", "salary" : 5000 }
{ "empID" : 113, "name" : "afroz", "design" : "developer", "salary" : 1500 }
{ "empID" : 114, "name" : "manish", "design" : "developer", "salary" : 10000 }
{ "empID" : 115, "name" : "sanket", "design" : "developer", "salary" : 15000 }
{ "empID" : 116, "name" : "abhijeet", "design" : "developer", "salary" : 25000 }
> db.emp.ensureIndex({empID:1})
{
  "createdCollectionAutomatically" : false,
  "numIndexesBefore" : 1,
  "numIndexesAfter" : 2,
  "ok" : 1
}
> db.emp.getIndexes()
[
  {
    "v" : 1,
    "key" : {
      "_id" : 1
    },
    "name" : "_id_",
    "ns" : "employee.emp"
  },
  {
    "v" : 1,
    "key" : {
      "empID" : 1
    },
    "name" : "empID_1",
    "ns" : "employee.emp"
  }
]
> db.emp.find()
{ "_id" : ObjectId("59a6427fd8da53a4381c8ea6"), "empID" : 111, "name" : "pratik", "design" : "ceo", "salary" : 3000 }
{ "_id" : ObjectId("59a64299d8da53a4381c8ea7"), "empID" : 112, "name" : "jatin", "design" : "manager", "salary" : 5000 }
{ "_id" : ObjectId("59a642a8d8da53a4381c8ea8"), "empID" : 113, "name" : "afroz", "design" : "developer", "salary" : 1500 }

```

```

{ "_id" : ObjectId("59a642d1d8da53a4381c8ea9"), "empID" : 114, "name" : "manish", "design" :
"developer", "salary" : 10000 }
{ "_id" : ObjectId("59a642ead8da53a4381c8eaa"), "empID" : 115, "name" : "sanket", "design" : "developer",
"salary" : 15000 }
{ "_id" : ObjectId("59a642ffd8da53a4381c8eab"), "empID" : 116, "name" : "abhijeet", "design" :
"developer", "salary" : 25000 }
> db.emp.ensureIndex({name:1})
{
  "createdCollectionAutomatically" : false,
  "numIndexesBefore" : 2,
  "numIndexesAfter" : 3,
  "ok" : 1
}
> db.emp.getIndexes()
[
  {
    "v" : 1,
    "key" : {
      "_id" : 1
    },
    "name" : "_id_",
    "ns" : "employee.emp"
  },
  {
    "v" : 1,
    "key" : {
      "empID" : 1
    },
    "name" : "empID_1",
    "ns" : "employee.emp"
  },
  {
    "v" : 1,
    "key" : {
      "name" : 1
    },
    "name" : "name_1",
    "ns" : "employee.emp"
  }
]
> db.emp.dropIndex({name:1})
{ "nIndexesWas" : 3, "ok" : 1 }
> db.emp.getIndexes()
[
  {
    "v" : 1,
    "key" : {
      "_id" : 1
    },
    "name" : "_id_",
    "ns" : "employee.emp"
  },
  {
    "v" : 1,
    "key" : {
      "empID" : 1
    },

```

```
        "name" : "empID_1",  
        "ns" : "employee.emp"  
    }  
]>
```

**Conclusion:** The Aggregation and Indexing in MongoDB is successfully understood.

## EXPERIMENT NO. B -4

### Map Reduce in MONGODB

**Title:** - Implement Map reduces operation with suitable example using MongoDB.

**Objective:** Understand map reduce operations in MongoDB

**Prerequisites:**

Basics of mongodb.

**Theory:** 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.

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.

Consider the following map-reduce operation:

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

**Syntax:**

\*\*\*\*\* MAP REDUCE \*\*\*\*\*

```
function mapf() {
  // 'this' holds current document to inspect
  emit(key, value);
}
```

```
function reducef(key,value_array) {
  return reduced_value;
}
```

```
db.mycollection.mapReduce(mapf, reducef[, options])
```

```
options
{[query : <query filter object>]
[, sort : <sort the query. useful for optimization>]
[, limit : <number of objects to return from collection>]
[, out : <output-collection name>]
[, keeptemp: <true|false>]
[, finalize : <finalizefunction>]
[, scope : <object where fields go into javascript global scope >]
[, verbose : true]}
```

Exmple:

```
> use VBK
switched to db VBK
> show tables;
Ass15
MRExample
bank8
bankMRE
bankMRE1
student
system.indexes
tea
users
> db.createCollection("TE_Marks")
{ "ok" : 1 }
> db.TE_Marks.insert({RollNo:1,Name:"ABC",Subject:"DMSA",Mark:12})
WriteResult({ "nInserted" : 1 })
> db.TE_Marks.insert({RollNo:1,Name:"ABC",Subject:"OSD",Mark:13})
WriteResult({ "nInserted" : 1 })
> db.TE_Marks.insert({RollNo:1,Name:"ABC",Subject:"DCWSN",Mark:15})
WriteResult({ "nInserted" : 1 })
> db.TE_Marks.insert({RollNo:1,Name:"ABC",Subject:"TOC",Mark:10})
WriteResult({ "nInserted" : 1 })
> db.TE_Marks.insert({RollNo:2,Name:"PQR",Subject:"DMSA",Mark:18})
WriteResult({ "nInserted" : 1 })
> db.TE_Marks.insert({RollNo:2,Name:"PQR",Subject:"OSA",Mark:13})
WriteResult({ "nInserted" : 1 })
> db.TE_Marks.insert({RollNo:2,Name:"PQR",Subject:"DCWSN",Mark:12})
WriteResult({ "nInserted" : 1 })
> db.TE_Marks.insert({RollNo:2,Name:"PQR",Subject:"TOC",Mark:13})
WriteResult({ "nInserted" : 1 })
> db.TE_Marks.find({})
{ "_id" : ObjectId("540ed8558ce9c9c60ab58273"), "RollNo" : 1, "Name" : "ABC", "Subject" : "DMSA", "Mark" : 12 }
{ "_id" : ObjectId("540ed8658ce9c9c60ab58274"), "RollNo" : 1, "Name" : "ABC", "Subject" : "OSD", "Mark" : 13 }
{ "_id" : ObjectId("540ed8718ce9c9c60ab58275"), "RollNo" : 1, "Name" : "ABC", "Subject" : "DCWSN", "Mark" : 15 }
{ "_id" : ObjectId("540ed87e8ce9c9c60ab58276"), "RollNo" : 1, "Name" : "ABC", "Subject" : "TOC", "Mark" : 10 }
{ "_id" : ObjectId("540ed8988ce9c9c60ab58277"), "RollNo" : 2, "Name" : "PQR", "Subject" : "DMSA", "Mark" : 18 }
{ "_id" : ObjectId("540ed8a18ce9c9c60ab58278"), "RollNo" : 2, "Name" : "PQR", "Subject" : "OSA", "Mark" : 13 }
{ "_id" : ObjectId("540ed8ac8ce9c9c60ab58279"), "RollNo" : 2, "Name" : "PQR", "Subject" : "DCWSN", "Mark" : 12 }
{ "_id" : ObjectId("540ed8bd8ce9c9c60ab5827a"), "RollNo" : 2, "Name" : "PQR", "Subject" : "TOC", "Mark" : 13 }
> var Mapfunction = function(){emit(this.RollNo,this.Mark)}
> var Reducefunction = function(key,values){return Array.sum(values)}
> db.TE_Marks.mapReduce(Mapfunction,Reducefunction,{ 'out':'MR_Result' })
{
  "result" : "MR_Result",
  "timeMillis" : 311,
  "counts" : {
    "input" : 8,
    "emit" : 8,
    "reduce" : 2,
    "output" : 2
  },
  "ok" : 1,
}
```

> show tables;

Ass15  
MRExample  
MR\_Result  
TE\_Marks  
bank8  
bankMRE  
bankMRE1  
student  
system.indexes  
tea  
users  
> db.MR\_Result.find({ })  
{ "\_id" : 1, "value" : 50 }  
{ "\_id" : 2, "value" : 56 }

// Average can be calculated by using...  
> var Reducefunction = function(key,values){return Array.avg(values)}

**Conclusion:** We understood the concept of Map Reduce in MongoDB.

## **EXPERIMENT NO. B -5**

### **Basic 5 Operations in MONGODB**

**Title:** - Implement Map reduces operation with suitable example using MongoDB.

**Objective:** Understand basic operations in MongoDB

**Prerequisites:**

Basics of mongodb.

**Theory:** 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.

Example:

```
> show databases;
VBK      0.078GB
admin    (empty)
local    0.078GB
newsletter 0.078GB
```

```
> use COEM // Created new Database named as "COEM"
switched to db COEM
```

```
> db.createCollection("TE") // Created new Collection named as "TE"
{ "ok" : 1 }
```

```
> db.TE.insert({Roll:1,Name:"ABC",Address:"Pune",Per:76})
WriteResult({ "nInserted" : 1 })
```

```
> db.TE.insert({Roll:2,Name:"PQR",Address:"Pune",Per:75})
WriteResult({ "nInserted" : 1 })
```

```
> db.TE.insert({Roll:3,Name:"LMN",Address:"Hadapsar",Per:70})
WriteResult({ "nInserted" : 1 })
```

```
> db.TE.find({})
{ "_id" : ObjectId("541963be2741c7552caef0a9"), "Roll" : 1, "Name" : "ABC", "Address" : "Pune", "Per" : 76 }
{ "_id" : ObjectId("541963cb2741c7552caef0aa"), "Roll" : 2, "Name" : "PQR", "Address" : "Pune", "Per" : 75 }
{ "_id" : ObjectId("541963dc2741c7552caef0ab"), "Roll" : 3, "Name" : "LMN", "Address" : "Hadapsar", "Per" : 70 }
```

```
> db.TE.find({})
{ "_id" : ObjectId("541963be2741c7552caef0a9"), "Roll" : 1, "Name" : "ABC", "Address" : "Pune", "Per" : 76 }
{ "_id" : ObjectId("541963cb2741c7552caef0aa"), "Roll" : 2, "Name" : "PQR", "Address" : "Pune", "Per" : 75 }
{ "_id" : ObjectId("541963dc2741c7552caef0ab"), "Roll" : 3, "Name" : "LMN", "Address" : "Hadapsar", "Per" : 70 }
```

```
> db.TE.update({Roll:2},{ $set:{Name:"Wagholi"}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

```
> db.TE.find({})
{ "_id" : ObjectId("541963be2741c7552caef0a9"), "Roll" : 1, "Name" : "ABC", "Address" : "Pune", "Per" : 76 }
```



```

{ "_id" : ObjectId("541963cb2741c7552caef0aa"), "Roll" : 2, "Name" : "Wagholi", "Address" : "Pune", "Per" : 75
}
{ "_id" : ObjectId("541963dc2741c7552caef0ab"), "Roll" : 3, "Name" : "LMN", "Address" : "Hadapsar", "Per" :
70 }

> db.TE.remove({Roll:3})
WriteResult({ "nRemoved" : 1 })

> db.TE.find({})
{ "_id" : ObjectId("541963be2741c7552caef0a9"), "Roll" : 1, "Name" : "ABC", "Address" : "Pune", "Per" : 76 }
{ "_id" : ObjectId("541963cb2741c7552caef0aa"), "Roll" : 2, "Name" : "Wagholi", "Address" : "Pune", "Per" : 75
}

> db.TE.drop()
true

> db.TE.find({})

> show collections
system.indexes

> show databases;
COEM      0.078GB
VBK       0.078GB
admin     (empty)
local     0.078GB
newsletter 0.078GB

> db.dropDatabase("COEM")
2014-09-17T16:17:18.278+0530 dropDatabase doesn't take arguments at src/mongo/shell/db.js:141

> db.dropDatabase()
{ "dropped" : "COEM", "ok" : 1 }

> show databases;
VBK       0.078GB
admin     (empty)
local     0.078GB
newsletter 0.078GB

```

**Conclusion:** We understood the basic operations in MongoDB.