Dr. D. Y. Patil Educational Federation's

Dr. D. Y. Patil College of Engineering & Innovation

Approved by AICTE, Affilated to SPPU Pune

Department of Computer Engineering
(A.Y. 2023-24)

Class: TE

Subject Name: Database Management Systems Laboratory (DBMS Lab Manual)

Subject Code: 310246

Faculty Name: Mr. Sunil Kumar Yadav

Roll No:

Name of student:

Class TE **Div:** **Batch:**

Index

Sr.No.	Title	Conducted Date	Submission Date/Sign
1	Propose a Conceptual Design using ER features using tools like ERDplus, ER Win etc. Convert the ER diagram into tables on paper and propose a normalise Relational data model.		
2	Write SQL queries using Insert, Select, Update, delete with operators,functions, and set operator etc. Use of SQL objects such as Table, View, Index, Sequence, Synonym, different constraints etc.		
3	Write SQL queries using Insert, Select, Update, delete with operators,functions, and set operator etc. Use of SQL objects such as Table, View, Index, Sequence, Synonym, different constraints etc.		
4	Write the SQL queries using of concepts like all types of Join, Sub- Query and View		
5	Write the SQL queries using of concepts like all types of Join, Sub- Query and View		
6	Write a Unnamed PL/SQL code block for given requirements (use ofControl structure and Exception handling is mandatory)		
7	Write a Named PL/SQL Block for given requirements using PL/SQLStored Procedure and Stored Function.		
8	Write a Named PL/SQL Block for given requirements using PL/SQLStored Procedure and Stored Function.		
9	Write a PL/SQL code block using Cursors (All types: Implicit, Explicit,Cursor FOR Loop, Parameterized Cursor)		
10	Write a PL/SQL code block using Cursors (All types: Implicit, Explicit,Cursor FOR Loop, Parameterized Cursor)		
11	Write a PL/SQL Code block using Database Trigger (All Types: Rowlevel and Statement level triggers, Before and After Triggers).		
12	Write a PL/SQL Code block using Database Trigger (All Types: Rowlevel and Statement level triggers, Before and After Triggers).		

13	Write a program to implement MYSQL/Oracle database connectivitywith any front-end language to implement Database navigation operations (add, delete, edit etc)		
14	Write the MongoDB Queries using CRUD operations. (Use CRUDoperations, SAVE method, logical operators etc)		
15	Write the MongoDB Queries using aggregation and indexing withsuitable example using MongoDB.		
16	Implement Map reduces operation with suitable example usingMongoDB.		
17	Write a program to implement MongoDB database connectivity withany front-end language to implement Database navigation operations (add, delete, edit etc)		

Subject Teacher sign **HOD sign**

Group A: Assignment No 1

Title of Assignment: ER Modeling and Normalization

Assignment Name: - Propose a Conceptual Design using ER features using tools like ERD plus, ER Win etc. (Identifying entities, relationships between entities, attributes, keys, cardinalities, generalization, specialization etc.) Convert the ER diagram into relational tables and normalize Relational data model.

Theory: -

Prerequisite: Basics of RDBMS.

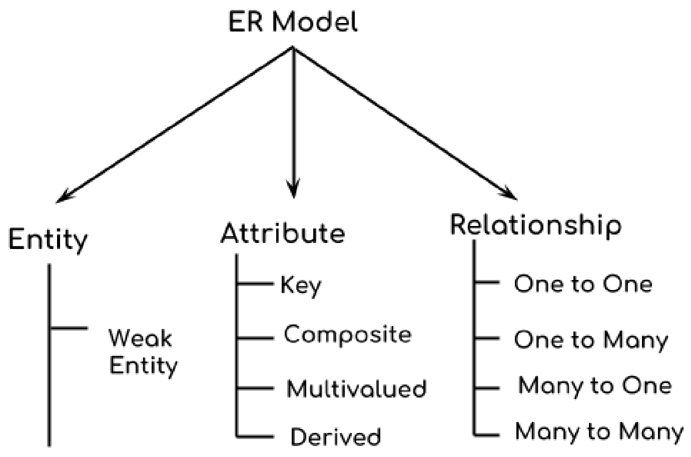
Objective:

- To learn and understand the concept offer Model and steps to Design ER Model and to ER Model into table.

New Concepts:

Entity-Relationship model is used in the conceptual design of a database (☞ conceptual level, conceptual schema). A database schema in the ER model can be represented pictorially (Entity-Relationship diagram)

Components of ER Diagram:



1> Entity: real-world object or thing with an independent existence and which is distinguishable from other objects. An entity is an object or component of data. An entity is represented as rectangle in an ER diagram.

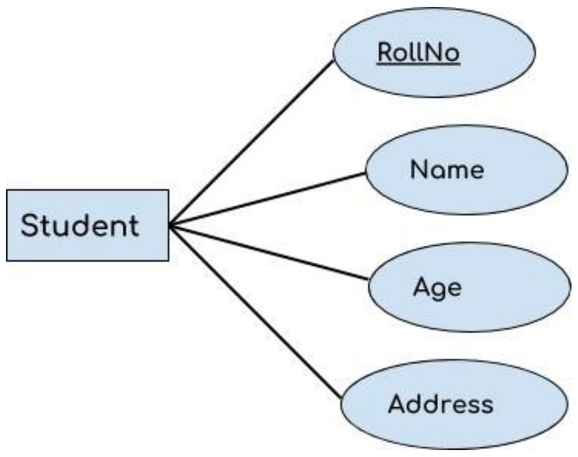
Examples are a person, car, customer, product, gene, book etc.

2> Attributes: an entity is represented by a set of attributes . An attribute describes the property of an entity. e.g., name, age, salary, price etc. Attribute values that describe each entity become a major part of the data eventually stored in a database.

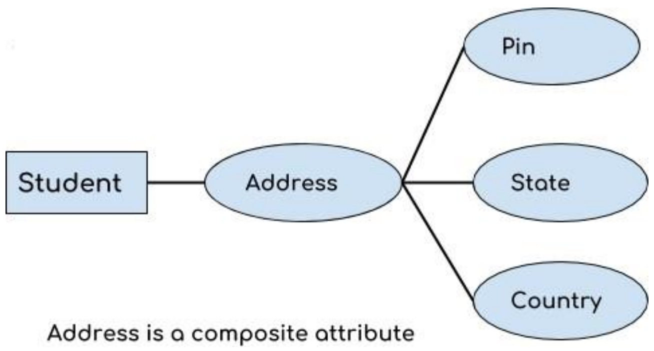
An attribute is represented as Oval in an ER diagram. There are four types of attributes:

1. Key attribute
2. Composite attribute
3. Multivalued attribute
4. Derived attribute

1. Key attribute:
ER diagram key attribute: A key attribute can uniquely identify an entity from an entity set. For example, student roll number can uniquely identify a student from a set of students. Key attribute is represented by oval same as other attributes however the text of key attribute is underlined.



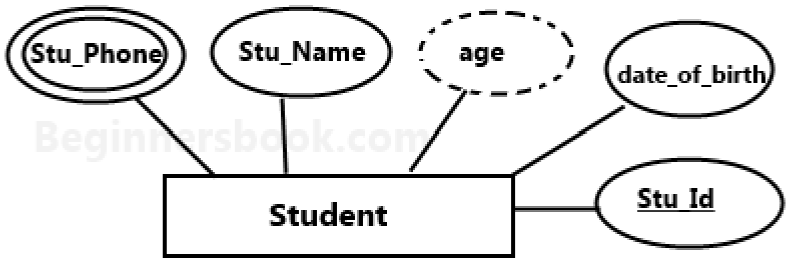
2. Composite attribute:
ER diagram composite attribute: An attribute that is a combination of other attributes is known as composite attribute. For example, In student entity, the student address is a composite attribute as an address is composed of other attributes such as pin code, state, country.



3. Multivalued attribute:
An attribute that can hold multiple values is known as multivalued attribute. It is represented with double ovals in an ER Diagram. For example – A person can have more than one phone numbers so the phone number attribute is multivalued.

4. Derived attribute:
A derived attribute is one whose value is dynamic and derived from another attribute. It is represented by dashed oval in an ER Diagram. For example – Person age is a derived attribute as it changes over time and can be derived from another attribute (Date of birth).

Multivalued and derived attribute
E-R diagram with multivalued and derived attributes:



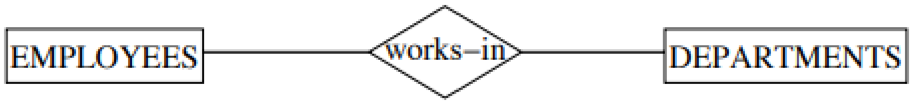
- Rectangles represent entity types
- Ellipses represent attributes
- Diamonds represent relationship types
- Lines link attributes to entity types and entity types to relationship types
- Primary key attributes are underlined
- Empty Circle at the end of a line linking an attribute to an entity type represents an optional (null) attribute
- Double Ellipses represent multi-valued attributes

3> Relationship

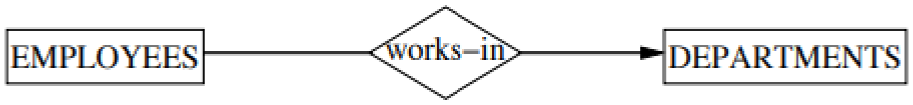
A relationship is represented by diamond shape in ER diagram, it shows the relationship among entities. There are four types of relationships:
1. One to One
2. One to Many
3. Many to One
4. Many to Many

Many-To-Many (default)

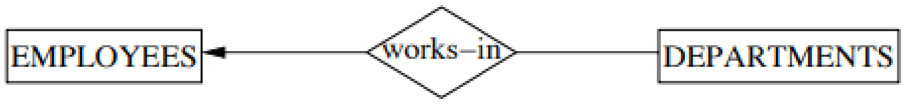
Meaning: An employee can work in many departments (≥ 0),and a department can have several employees



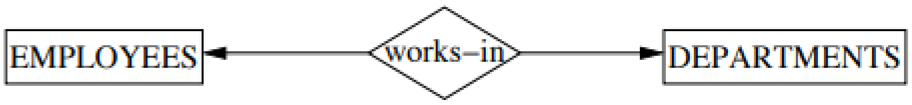
Many-To-One
Meaning: An employee can work in at most one department (≤ 1), and a department can have several employees.



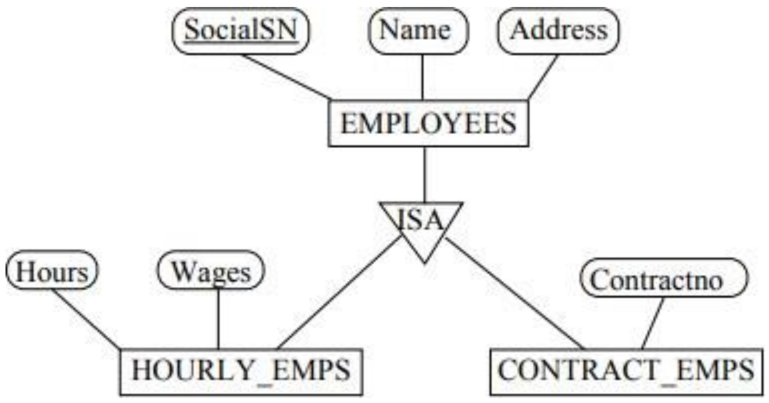
One-To-Many
Meaning: An employee can work in many departments (≥ 0), but a department can have at most one employee.



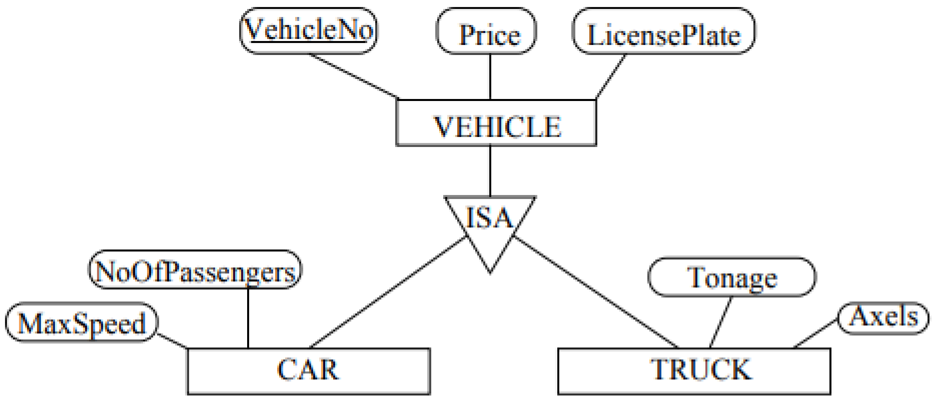
One-To-One
Meaning: An employee can work in at most one department, and a department can have at most one employee.



Specialization
• Process of defining a set of subclasses of an entity type (top-down)
HOURLY EMPS is a subclass of EMPLOYEES and thus inherits its attributes and relationships (same for CONTRACT EMPS).



Generalization:
• Reverse process of specialization (bottom-up); identify common features of entity types and generalize them into single superclass (including primary key!)



Steps in Designing an Entity-Relationship Schema

[Step 1] Identify entity types (entity type vs. attribute)
[Step 2] Identify relationship types
[Step 3] Identify and associate attributes with entity and relationship types
[Step 4] Determine attribute domains
[Step 5] Determine primary key attributes for entity types
[Step 6] Associate (refined) cardinality ratio(s) with relationship types

[Step 7] Design generalization/specialization hierarchies
including constraints (includes natural language statements
as well)

Translation of ER Schema into Tables

• An ER schema can be represented by a collection of tables which represent contents of the database (instance).

• Primary keys allow entity types and relationship types to be expressed uniformly as tables.

• For each entity and relationship type, a unique table can be derived which is assigned the name of the corresponding entity or relationship type.

• Each table has a number of columns that correspond to the attributes and which have unique names. An attribute of a table has the same domain as the attribute in the ER schema.

• Translating an ER schema into a collection of tables is the basis for deriving a relational database schema from an ER diagram.

ERD Plus: A database modeling tool for creating Entity Relationship Diagrams, Relational Schemas, Star Schemas, and SQL DDL statements.
https://erdplus.com/standalone

Conclusion: So in this way ,we have study the ER Diagram successfully.

Group A: Assignment No 2a

Title of Assignment: SQL DDL Statements

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

Theory: -

Prerequisite: Basics of RDBMS.

Objective:

- To learn and understand the concept and DDL queries of SQL Table,View and Index using MYSQL using 2-tier .

New Concepts:

Introduction to MySQL Table DDL Commands.

CREATE Table

Use a **`CREATE TABLE`** statement to specify the layout of your table:

```
mysql> CREATE TABLE Student (roll_no int (3) Primary key
auto_increment , name varchar(20) , age int(3));
```

`VARCHAR` is a good choice for the `name`, `owner`, and `species` columns because the column values vary in length. The lengths in those column definitions need not all be the same, and need not be `20`. You can normally pick any length from `1` to `65535`, whatever seems most reasonable to you.

DESCRIBE Table

To verify that your table was created the way you expected, use a `DESCRIBE` statement:

```
mysql> DESC Student;
```

```
+------------------+----------------------------+---------+--------+------------+----------------------+
| Field            | Type                       | Null    | Key    | Default    | Extra                |
+------------------+----------------------------+---------+--------+------------+----------------------+
| roll_no          | int  (3)                   | NO      | PRI    | NULL       | auto_increment       |
| name             | varchar(20)                | NO      |        | NULL       |                      |
| age              | int(3)                     | NO      |        | NULL       |                      |
+------------------+----------------------------+---------+--------+------------+----------------------+
```

<u>Constraints</u>

- **NOT NULL** - a value is required

  Example

  ```
  Create table stud (name varchar (20) not null);
  ```

- **UNIQUE -** the attribute value must be unique in the table

Example

```
Create table stud (rollno number(4) constraint uk1 unique key);
```

- **PRIMARY KEY** - unique and used to refer to tuples

  Example

```
Create table stud (rollno number(4)constraint pk1 primary key)
```

- **FOREIGN KEY** – It refers to the another column which is primary key

  Syntax:

```
Create table table_name(Attr_name Attr_type(size)references     table_name(attr_name))
```

  Example

```
Create table stud1 (rollno number(4) references stud(rollno));
```

- **CHECK (condition)** - for general constraints on a table's contents.

  Example

```
CHECK (rollno BETWEEN 1 AND 60)
```

You can also add constraint after table creation using alter table option

Eg Alter table stud add constraint **prk1** primary key(rollno);

You can also drop constraint using Drop command & name of constraint

Eg Drop constraint **prk1**;

ALTER Table

The ALTER TABLE statement is used to add, delete, or modify columns in an existing table

Syntax

```
Alter table table_name
add/modify/drop
column_name data_type(size)
```

Example

Following example shows use of Alter command. First example add new column Address in table and second example modify in size of age column. Third example shows how to drop column age.

```
1> Alter table Student add Address varchar(30);
2> Alter table Student  modify age int(5);
3> Alter table Student drop column age;
```

DROP Table

The DROP TABLE statement is used to delete a table.

```
DROP TABLE table_name
```

TRUNCATE TABLE

What if we only want to delete the data inside the table, and not the table itself?

Then, use the TRUNCATE TABLE statement:

```
TRUNCATE TABLE table_name
```

VIEW in MySQL

CREATE VIEW Statement

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

SQL CREATE VIEW Syntax

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

SQL Dropping a View

You can delete a view with the DROP VIEW command.

```
DROP VIEW view_name
```

INDEX in MySQL

Create Index Statement

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

Syntax to create Index:

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

```
ALTER TABLE table_name ADD INDEX index_name (column_name);
```

Show Index Information

```
show index from table_name
```

```
Table    | Non_unique | Key_name | Seq_in_index | Column_name | Collation |
Cardinality | Sub_part | Packed | Null | Index_type | Comment
```

Drop Index

```
ALTER TABLE table_name DROP INDEX index_name
```

Group A: Assignment No 2b

Title of Assignment: SQL DML Statements

Assignment Name: - Write at least 10 SQL queries on the suitable database application using SQL DML statements.

Theory: -

Prerequisite: SQL DDL queries.

Objective:

- To learn and understand the DML queries of SQL using MYSQL

New Concepts:

3.1 Introduction to MySQL DML (Data Manipulation Language)Commands.

1>. **INSERT** tuples into a relation:

INSERT INTO tablename

VALUES (attr1_value, attr2_value, ... attrn_value)

Where every attribute in the table is given a value

INSERT INTO tablename (attrx, ... , attry)

VALUES (attrx_value, ... , attry_value)

which leaves other attributes as NULL. e.g.

Example:

1. **Insert into Stud values (1,'Pooja', 'Pune')**
2. **Insert into Stud (rollno,name) values (2, 'Amit')**

The output of "Stud" table after inserting above 2 rows

rollno	**Name**	**Address**
1	Pooja	Pune
2	Amit	

2.> **DELETE** tuples from a relation:

DELETE FROM tablename **WHERE** condition

removes zero, one or many tuples;The WHERE clause is optional; if it is left out all tuples are removed, but the table still exists; it is empty.

DELETE FROM Stud **WHERE** Address = 'Pune'; (Only delete roes that satisfy the condition)

DELETE FROM Stud; (Delete all rows)

3> UPDATE of one or more selected tuples

```
UPDATE tablename
SET attrx = new_value,
WHERE condition;
```

Example:

```
Update Stud Set Name= 'Shruti' where rollno=1
```

4>SELECT statement retrieves data from table

```
SELECT attribute_list
FROM table_list
WHERE condition;
```

The **attributes** are those you want to see in the result,

the **tables** are those required for the query,

the **condition** is a boolean expression that specifies which tuples are to be retrieved by the query.

Example:

```
Select Name, Address from Stud;
```

3.2 RETRIEVAL WITH ORDERING:

The ORDER BY keyword is used to sort the result-set by a specified column.
The ORDER BY keyword sort the records in ascending order by default.
If you want to sort the records in a descending order, you can use the DESC keyword.

SQL ORDER BY Syntax

```
SELECT column_name(s)
FROM table_name
ORDER BY column_name(s) ASC|DESC
```

3.3 The LIKE Operator

The LIKE operator is used to search for a specified pattern in a column.

SQL LIKE Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name LIKE pattern
```

SQL Wildcards

SQL wildcards can substitute for one or more characters when searching for data in a database.

SQL wildcards must be used with the SQL LIKE operator.

With SQL, the following wildcards can be used:

Wildcard	Description
%	A substitute for zero or more characters
_	A substitute for exactly one character

Example:

The "Persons" table:

P_Id	LastName	Address
1	Heena	Pune
2	Savita	Pune
3	Sarika	Bombay

We use the following SELECT statement:

```
SELECT  LastName  from Person where LastName like ‘S%’ ;
```

The result-set will look like this:

LastName
Savita
Sarika

3.4 DISTINCT Statement

In a table, some of the columns may contain duplicate values. This is not a problem, however, sometimes you will want to list only the different (distinct) values in a table.

The DISTINCT keyword can be used to return only distinct (different) values.

SQL SELECT DISTINCT Syntax

```
SELECT DISTINCT column_name(s) FROM table_name
```

The "Persons" table:

P_Id	LastName	FirstName	Address
1	Hansen	Ola	Pune
2	Svendson	Tove	Pune
3	Pettersen	Kari	Bombay

We use the following SELECT statement:

```
SELECT distinct(Address) from Person
```

The result-set will look like this:

Address
Pune
Bombay

3.5 The IN Operator

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

SQL IN Syntax

```
SELECT column_name(s)  FROM table_name
WHERE column_name IN (value1,value2,...)
```

IN Operator Example

The "Persons" table:

P_Id	LastName	FirstName	Address
1	Hansen	Ola	Pune
2	Svendson	Tove	Nashik
3	Pettersen	Kari	Bombay

Now we want to select the persons with a last name equal to "Hansen" or "Pettersen" from the table above.

We use the following SELECT statement:

```
SELECT * FROM Persons
WHERE Address IN ('Pune','Bombay')
```

The result-set will look like this:

P_Id	LastName	FirstName	Address
1	Hansen	Ola	Pune
3	Pettersen	Kari	Bombay

3.6 All tuples/ all attributes:

If you leave out the WHERE clause you get **all the tuples** in the specified tables.
Use '*' as the attribute list to specify **all the attributes** of the specified tables.
SELECT * **FROM** table_name ;

3.7 NULL Operator

To test if a value is NULL you cannot use =, because every NULL is considered distinct from every other one. Must use "IS NULL" or "IS NOT NULL"
e.g. names of all employees who do not have supervisors

SELECT * **FROM** table_name
WHERE Attr_name **IS NULL;**

Example

The "Persons" table:

P_Id	LastName	FirstName	Address
1	Hansen	Ola	Pune
2	Svendson	Tove	
3	Pettersen	Kari	Bombay

We use the following SELECT statement:

```
SELECT * FROM Persons
WHERE Address is null;
```

Output will be:

P_Id	LastName	FirstName	Address
2	Svendson	Tove	

3.8 Aggregate Functions

SQL Aggregate Functions

SQL aggregate functions return a single value, calculated from values in a column.

Useful aggregate functions:

- AVG() - Returns the average value
- COUNT() - Returns the number of rows
- MAX() - Returns the largest value
- MIN() - Returns the smallest value
- SUM() - Returns the sum

SQL AVG() Syntax

```
SELECT AVG(column_name) FROM  table_name
```

SQL AVG() Example

```
SELECT AVG(Marks) FROM  Stud
```

SQL COUNT(column_name) Syntax

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

```
SELECT COUNT(column_name) FROM table_name
```

SQL COUNT(*) Syntax

The COUNT(*) function returns the number of records in a table:

```
SELECT COUNT(*) FROM table_name
```

3.9 GROUP BY CLAUSE

This clause allows the rows of a table to be grouped by the value of some attribute(s); the SELECT clause is applied to each group: The GROUP BY statement is used in conjunction with the aggregate functions to group the result-set by one or more columns.

SQL GROUP BY Syntax

```
SELECT column_name, aggregate_function(column_name)
FROM table_name
WHERE column_name operator value
GROUP BY column_name
```

SQL GROUP BY Example

We have the following "Orders" table:

O_Id	OrderDate	OrderPrice	Customer
1	2008/11/12	1000	Hansen
2	2008/10/23	1600	Nilsen
3	2008/09/02	700	Hansen

4	2008/09/03	300	Hansen
5	2008/08/30	2000	Jensen
6	2008/10/04	100	Nilsen

Now we want to find the total sum (total order) of each customer.

We will have to use the GROUP BY statement to group the customers.

We use the following SQL statement:

```
SELECT Customer,SUM(OrderPrice) FROM Orders
GROUP BY Customer
```

The result-set will look like this:

Customer	SUM(OrderPrice)
Hansen	2000
Nilsen	1700
Jensen	2000

3.10 The HAVING Clause

The HAVING clause was added to SQL because the WHERE keyword could not be used with aggregate functions.

SQL HAVING Syntax

```
SELECT column_name, aggregate_function(column_name)
FROM table_name
WHERE column_name operator value
GROUP BY column_name
HAVING aggregate_function(column_name) operator value
```

SQL HAVING Example

We have the following "Orders" table:

O_Id	OrderDate	OrderPrice	Customer
1	2008/11/12	1000	Hansen
2	2008/10/23	1600	Nilsen
3	2008/09/02	700	Hansen
4	2008/09/03	300	Hansen
5	2008/08/30	2000	Jensen

6	2008/10/04	100	Nilsen

Now we want to find if any of the customers have a total order of less than 2000.

We use the following SQL statement:

```
SELECT Customer,SUM(OrderPrice) FROM Orders
GROUP BY Customer
HAVING SUM(OrderPrice)<2000
```

The result-set will look like this:

Customer	**SUM(OrderPrice)**
Nilsen	1700

Group A: Assignment No 3

Title of Assignment: SQL Queries – all types of Join, Sub-Query and View:

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

Theory: -

Prerequisite: Basic DDL and DML queries.

Objective:

- To learn and understand the concept and queries of SQL joints using MYSQL
- . To learn and understand the concept and queries of SQL Sun-Queries and joins using MYSQL

New Concepts:

Introduction to Joins: An **SQL JOIN** clause is used to combine rows from two or more tables, based on a common field between them. Before we continue with examples, we will list the types of JOIN you can use, and the differences between them.

- **JOIN**: Return rows when there is at least one match in both tables
- **LEFT JOIN**: Return all rows from the left table, even if there are no matches in the right table
- **RIGHT JOIN**: Return all rows from the right table, even if there are no matches in the left table
- **FULL JOIN**: Return rows when there is a match in one of the tables

SQL JOIN Syntax

```
SELECT column_name(s)
FROM table_name1,table_name2
WHERE table_name1.column_name=table_name2.column_name
```

SQL JOIN EXAMPLE

The "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

The "Orders" table:

O_Id	OrderNo	P_Id
1	77895	3
2	44678	3
3	22456	1
4	24562	1
5	34764	15

Now we want to list all the persons with any orders.

We use the following SELECT statement:

```
SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo
FROM Persons, Orders
WHERE Persons.P_Id=Orders.P_Id
```

The result-set will look like this:

LastName	FirstName	OrderNo
Hansen	Ola	22456
Hansen	Ola	24562
Pettersen	Kari	77895
Pettersen	Kari	44678

SQL LEFT JOIN KEYWORD

The LEFT JOIN keyword returns all rows from the left table (table_name1), even if there are no matches in the right table (table_name2).

SQL LEFT JOIN Syntax

```
SELECT column_name(s)
FROM table_name1
LEFT JOIN table_name2
ON table_name1.column_name=table_name2.column_name
```

SQL LEFT JOIN EXAMPLE

```
SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo
FROM Persons
LEFT JOIN Orders
ON Persons.P_Id=Orders.P_Id
```

The result-set will look like this:

LastName	FirstName	OrderNo
Hansen	Ola	22456
Hansen	Ola	24562
Pettersen	Kari	77895
Pettersen	Kari	44678
Svendson	Tove	

The LEFT JOIN keyword returns all the rows from the left table (Persons), even if there are no matches in the right table (Orders).

SQL RIGHT JOIN KEYWORD

The RIGHT JOIN keyword Return all rows from the right table (table_name2), even if there are no matches in the left table (table_name1).

SQL RIGHT JOIN Syntax

```
SELECT column_name(s)
FROM table_name1
```

```
RIGHT JOIN table_name2
ON table_name1.column_name=table_name2.column_name
```

SQL RIGHT JOIN EXAMPLE: Now we want to list all the orders with containing persons - if any, from the tables above.

```
SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo
FROM Persons
RIGHT JOIN Orders
ON Persons.P_Id=Orders.P_Id
```

The result-set will look like this:

LastName	FirstName	OrderNo
Hansen	Ola	22456
Hansen	Ola	24562
Pettersen	Kari	77895
Pettersen	Kari	44678
		34764

The RIGHT JOIN keyword returns all the rows from the right table (Orders), even if there are no matches in the left table (Persons).

SQL FULL JOIN KEYWORD

The FULL JOIN keyword return rows when there is a match in one of the tables.

SQL FULL JOIN Syntax

```
SELECT column_name(s)
FROM table_name1
FULL JOIN table_name2
ON table_name1.column_name=table_name2.column_name
```

SQL FULL JOIN EXAMPLE

Now we want to list all the persons and their orders, and all the orders with their persons.

```
SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo
FROM Persons
FULL JOIN Orders
ON Persons.P_Id=Orders.P_Id
```

The result-set will look like this:

LastName	FirstName	OrderNo
Hansen	Ola	22456
Hansen	Ola	24562
Pettersen	Kari	77895
Pettersen	Kari	44678
Svendson	Tove	
		34764

The FULL JOIN keyword returns all the rows from the left table (Persons), and all the rows from the right table (Orders). If there are rows in "Persons" that do not have matches in "Orders", or if there are rows in "Orders" that do not have matches in "Persons", those rows will be listed as well.

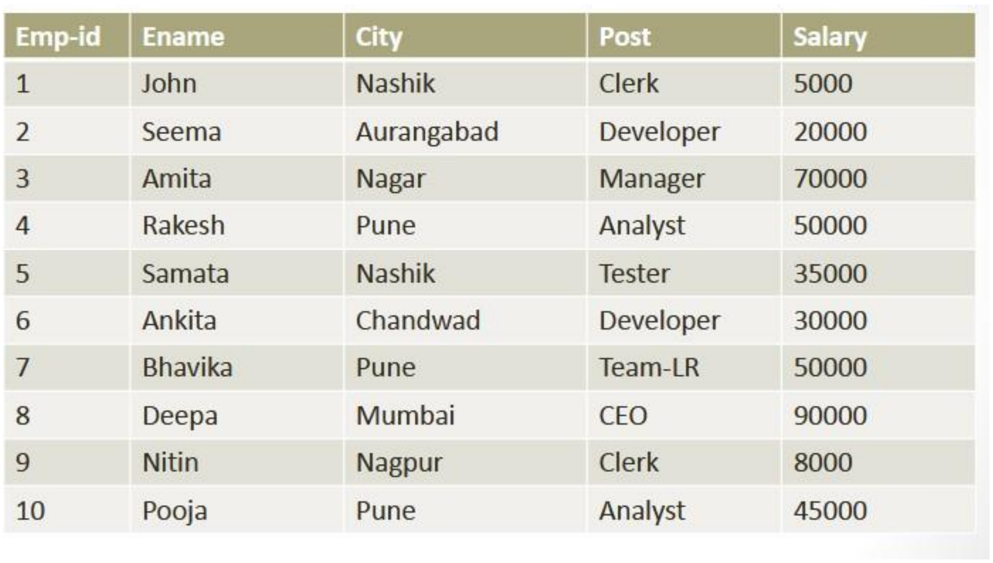
Types of Subqueries

Single Row Sub Query: Sub query which returns single row output. They mark the usage of single row comparison operators, when used in WHERE conditions.

Multiple row sub query: Sub query returning multiple row output. They make use of multiple row comparison operators like IN, ANY, ALL. There can be sub queries returning multiple columns also.

Correlated Sub Query: Correlated subqueries depend on data provided by the outer query.Thistype of subquery also includes subqueries that use the EXISTS operator to test the existence of data rows satisfying specified criteria.

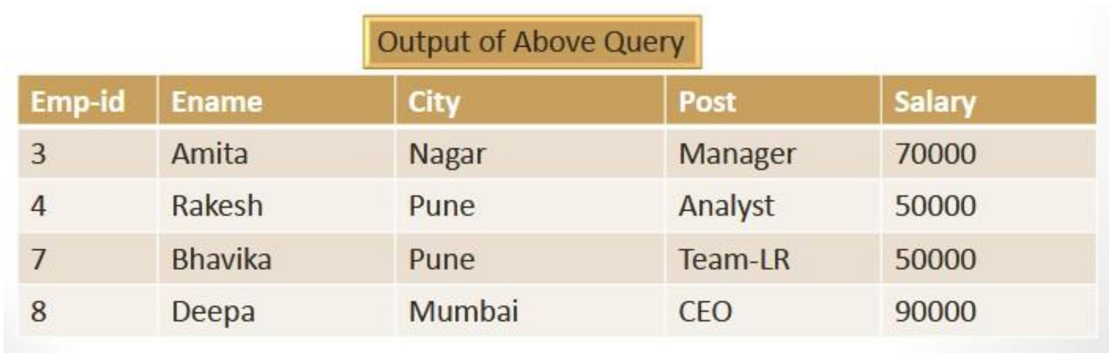
Example:



Emp-id	Ename	City	Post	Salary
1	John	Nashik	Clerk	5000
2	Seema	Aurangabad	Developer	20000
3	Amita	Nagar	Manager	70000
4	Rakesh	Pune	Analyst	50000
5	Samata	Nashik	Tester	35000
6	Ankita	Chandwad	Developer	30000
7	Bhavika	Pune	Team-LR	50000
8	Deepa	Mumbai	CEO	90000
9	Nitin	Nagpur	Clerk	8000
10	Pooja	Pune	Analyst	45000

Display the information of employees, paid more than 'pooja' from emptable

Select *from emp where salary > (select salaryfrom empwhere name='Pooja') ;



Output of Above Query

Emp-id	Ename	City	Post	Salary
3	Amita	Nagar	Manager	70000
4	Rakesh	Pune	Analyst	50000
7	Bhavika	Pune	Team-LR	50000
8	Deepa	Mumbai	CEO	90000

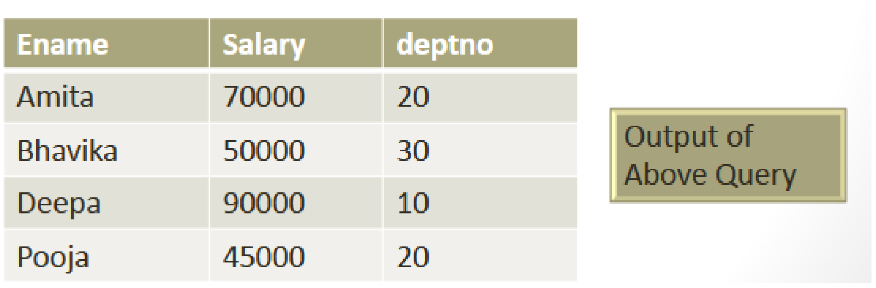
MySQL Subqueries -Multiple rows with ALL, ANY, IN operator

- [> ALL] More than the highest value returned by the subquery
- [< ALL] Less than the lowest value returned by the subquery
- [< ANY] Less than the highest value returned by the subquery
- [> ANY] More than the lowest value returned by the subquery
- [= ANY] Equal to any value returned by the subquery (same as IN)

>All Example-

Display the employee name, salary and department no of those employees whose salary is higher than all developers' salary.

SELECT Ename, salary, deptnoFROM EMPWHERE salary > All (SELECT salaryFROM empWhere post='Developer')



Ename	Salary	deptno
Amita	70000	20
Bhavika	50000	30
Deepa	90000	10
Pooja	45000	20

Output of Above Query

Group A: Assignment No 4 & 5

Title of Assignment: PL/SQL Procedures

Assignment Name: - 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, Date_of_Issue, 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.

OR

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

Theory: -

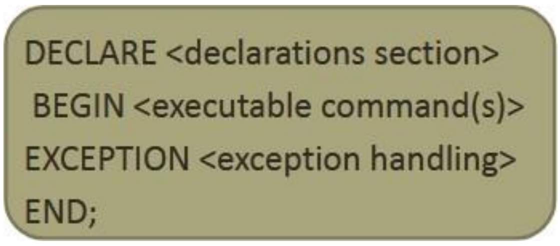
PL/SQL Introduction

•PL/SQL is a combination of SQL along with the procedural features of programming languages

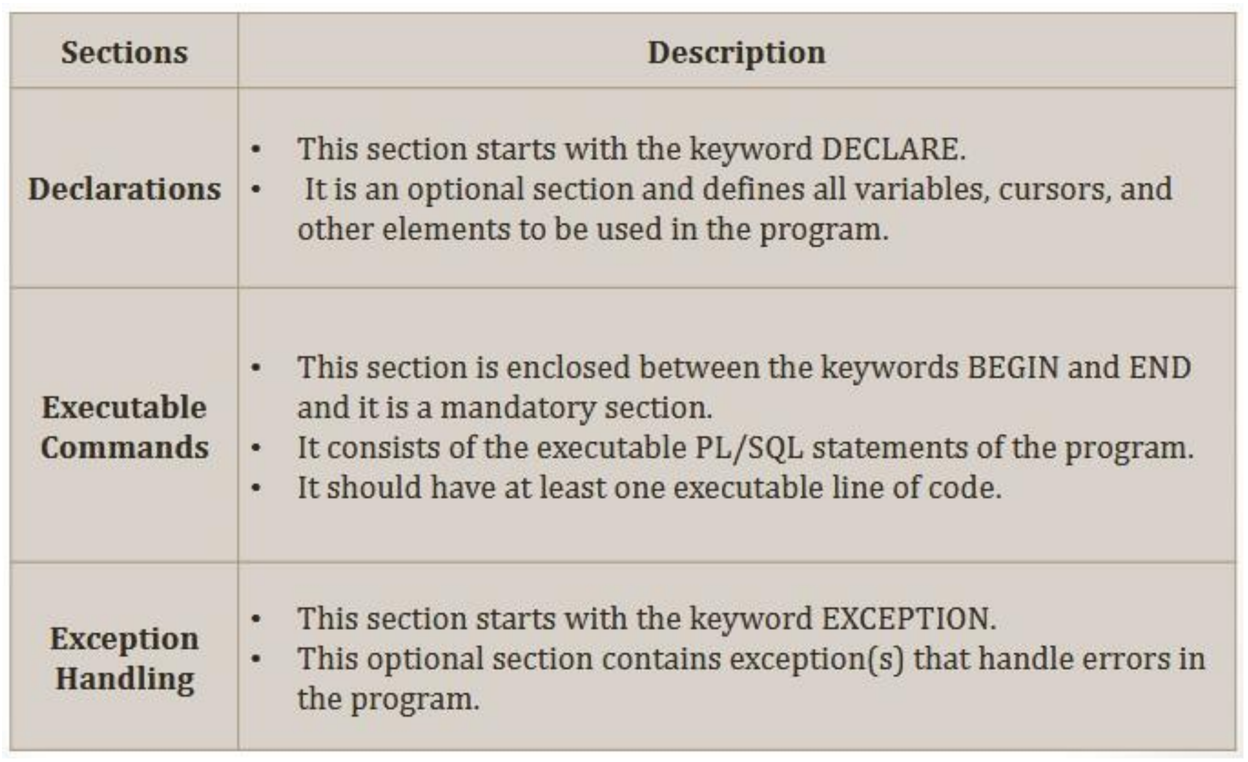
•Basic Syntax of PL/SQL which is a block-structured language; this means that the PL/SQL programs are divided and written in logical blocks of code. Each block consists of three sub-parts

•Every PL/SQL statement ends with a semicolon (;)

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



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



Sections	Description
Declarations	• This section starts with the keyword DECLARE. • It is an optional section and defines all variables, cursors, and other elements to be used in the program.
Executable Commands	• This section is enclosed between the keywords BEGIN and END and it is a mandatory section. • It consists of the executable PL/SQL statements of the program. • It should have at least one executable line of code.
Exception Handling	• This section starts with the keyword EXCEPTION. • This optional section contains exception(s) that handle errors in the program.

Anonymous blocks: Unnamed

Anonymous blocks are PL/SQL blocks which do not have any names assigned to them.

They need to be created and used in the same session because they will not be stored in the server as a database objects.

Since they need not to store in the database, they need no compilation steps.

They are written and executed directly, and compilation and execution happen in a single process.

Named blocks:

Named blocks are having a specific and unique name for them.They are stored as the database objects in the server. Since they are available as database objects, they can be referred to or used as long as it is present in the server.

Named blocks are basically of two types:

1. Procedure
2. Function

Stored Procedure Syntax

CREATE PROCEDURE sp_name([proc_parameter: [IN | OUT | INOUT] param_namedata_type])

Begin<Declare variable_namedata_type;>

<Control Statements/loops>

SQL executable statements;

End

Stored Procedure-Parameters

IN –is the default mode. When you define an IN parameter in a stored procedure, the calling program has to pass an argument to the stored procedure.

OUT –the value of an OUT parameter can be changed inside the stored procedure and its new value is passed back to the calling program

INOUT –an INOUT parameter is the combination of IN and OUT parameters. It means that the calling program may pass the argument, and the stored procedure can modify the INOUT parameter and pass the new value back to the calling program.

The IN parameter example

```
Mysql> DELIMITER  //
Mysql> CREATE PROCEDURE Allstud(IN SNameVARCHAR(25))
BEGIN
SELECT *FROM stud where Name=SName;
END
//
Mysql> DELIMITER ;
Mysql> call Allstud('Reena');
```

The IN and OUT parameter example

```
Mysql> CREATE PROCEDURE Allstud(IN Rno1 int,OUT SName VARCHAR)
BEGIN
SELECT Name into SName FROM stud where Rno=RNo1;
END
//
Mysql> call Allstud(2,@SName)//
Mysql> select @Sname
```

Date Related Functions required to solve the assignment

CURDATE() -•The CURDATE() function returns the current date.

Note:This function returns the current date as a "YYYY-MM-DD" format

•Example

•Return current date:

•SELECT CURDATE();

DATEDIFF()

The DATEDIFF() function returns the difference in days between two date values.

•Syntax

•DATEDIFF(date1, date2)

•Example

•Return the difference in days between two date values:

•SELECT DATEDIFF("2017-06-25", "2017-06-15");

Exception handling

Declaring a handler

•To declare a handler, you use thestatement as follows:

•DECLARE action HANDLER FOR condition_value statement;

•If a condition whose value matches the condition_value, MySQL will execute the statement and continue or exit the current code block based on the action .

•The action accepts one of the following values:

1. CONTINUE :the execution of the enclosing code block (BEGIN ... END) continues.
2. EXIT : the execution of the enclosing code block, where the handler is declared, terminates.

Group A: Assignment No : 6

Title of Assignment: PL/SQL Stored Function

Assignment Name: -.

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

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

- Stud_Marks(name, total_marks)
- Result(Roll,Name, Class)

Theory: -

What's a Stored Function?

If procedural programming is new to you, you may be wondering what the difference is between a Stored Procedure and a Stored Function. Not too much really.

A function always returns a result, and can be called inside an SQL statement just like ordinary SQL functions.

A function parameter is the equivalent of the IN procedure parameter, as functions use the RETURN keyword to determine what is passed back.

Stored functions also have slightly more limitations in what SQL statements they can run than stored procedures.

Syntax For Creating a Function

A standalone function is created using the **CREATE FUNCTION** statement. The simplified syntax for the **CREATE OR REPLACE PROCEDURE** statement is as follows −

```
CREATE [OR REPLACE] FUNCTION function_name
[(parameter_name [IN | OUT | IN OUT] type [, ...])]
RETURNS return_datatype
BEGIN
  < function_body >
END;
```

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 *RETURNS* clause specifies the data type you are going to return from the function.

- *function-body* contains the executable part.

To call Function: Select Fun_name(Parameter Value)

Example 1: Function for addition of two numbers.

- MySQL> Delimiter //   To set the delimiter
- MySQL>Create function Add1(a int, b int) returns int

```
Begin
 Declare int;
Set c= a+b;
Return c;
End;
//
```

- MySQL> Select Add1(10,20) ;   To run the Function

Example 2: Take rollno as input from user and pass to function, Function will return name of student belonging to that rollno.

- MySQL>Create function Rname(rno1 int) returns varchar(20)

```
Begin
Declare sname  varchar(20);
Select name into sname from Stud  where rno=rno1;
Return sname;
End;
//
```

- MySQL> Select Rname(1) ;   To run the Function

Example 3: Function to return grade depend on his/her marks.

```
MySQL>Create function Fgrade(rno1 int) returns varchar(20)
        Begin
         Declare  grade varchar(20);
         Declare mark1 int;
        Select mark into mark1 from Stud  where rno=rno1;
        If (mark1 >75) then
        Set grade= ‘Distinction’
        ElseIf  (mark1>=60 and mark1<75) then
        Set grade=‘First Class’
        ElseIf (marks <60) then
        Set grade=‘Pass Class’
        End if;
        Return grade;
        End;
        //
MySQL> Select Fgrade(1) ;                    To run the Function
```

Assignment to Perform:

- Write a Stored Procedure/Function namely proc_Grade for the categorization of student.
    - If marks scored by students in examination is <=1500 and marks>=990 then distinction category
    - If marks scored are between 989 and900 category is first class,
    - If marks 899 and 825 category is Higher Second Class
- Schema Required are:
    - Stud_Marks(name, total_marks)
    - Result(Roll,Name, Class

Group A: Assignment No 7

Title of Assignment: Cursors: (All types: Implicit, Explicit, Cursor FOR Loop, Parameterized Cursor)

Assignment Name: - 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.

Theory: -

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

To handle a result set inside a stored procedure, you use a cursor. A cursor allows you to iterate a set of rows returned by a query and process each row accordingly.

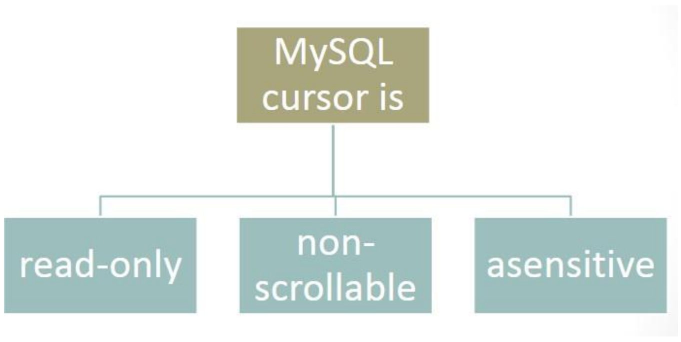
Types of Cursor

1>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. Implicit Cursor is associated with following DML Statements INSERT, UPDATE , DELETE

2>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.
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

Types of Cursor in MySQL

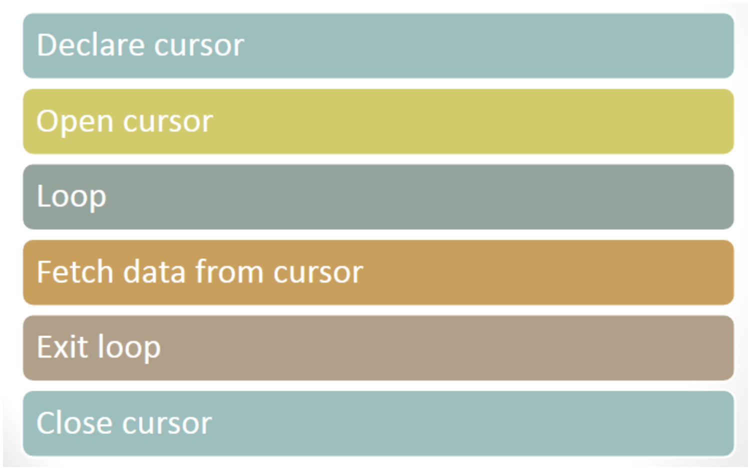


Read only: you cannot update data in the underlying table through the cursor

Non-scrollable: you can only fetch rows in the order determined by the SELECT statement. You cannot fetch rows in the reversed order. In addition, you cannot skip rows or jump to a specific row in the result set

Asensitive: there are two kinds of cursors: asensitive cursor and insensitive cursor. An asensitive cursor points to the actual data, whereas an insensitive cursor uses a temporary copy of the data. An asensitive cursor performs faster than an insensitive cursor because it does not have to make a temporary copy of data.

Step for Using Cursor



1. **DECLARE statement:** The cursor declaration must be after any variable declaration. •A cursor must always be associated with a SELECT statement

   Syntax

   DECLARE cursor_nameCURSOR FOR SELECT_statement;

2. **OPEN statement:** The OPEN statement initializes the result set for the cursor, therefore, you must call the OPEN statement before fetching rows from the result set.

   Syntax
   OPEN cursor_name;

3. **LOOP statement:** The LOOP depends on the careful placement of the LEAVE statement to terminate iteration.

   Syntax
   [label_name:]
   LOOPstatement_list
   END LOOP [label_name]

4. **FETCH statement:** Then, you use the FETCH statement to retrieve the next row pointed by the cursor and move the cursor to the next row in the result set.

   Syntax

   FETCH cursor_name INTO variables list;

5. **CLOSE statement:** Finally, you call the CLOSE statement to deactivate the cursor and release the memory associated with it

Syntax

CLOSE cursor_name;

6. **NOT FOUND handler**

   When working with MySQL cursor, you must also declare a NOT FOUND handler to handle the situation when the cursor could not find any row. Because each time you call the FETCH statement, the cursor attempts to read the next row in the result set. When the cursor reaches the end of the result set, it will not be able to get the data, and a condition is raised. The handler is used to handle this condition.

Syntax

```
DECLARE CONTINUE HANDLER FOR NOT FOUND SET exit_loop= TRUE
```

Cursor Example 1:

```
CREATE PROCEDURE cursor_proc2()
 BEGIN

   DECLARE id   VARCHAR(3);
   DECLARE name1 VARCHAR(20);

 -- this flag will be set to true when cursor reaches end of table
   DECLARE exit_loop BOOLEAN;
   -- Declare the cursor
   DECLARE c1 CURSOR FOR   SELECT rno, name FROM stud;

   -- set exit_loop flag to true if there are no more rows
   DECLARE CONTINUE HANDLER FOR NOT FOUND SET exit_loop = TRUE;
   -- open the cursor
 OPEN c1;

 -- start looping
   L1: LOOP

  -- read the id and name from next row into the variables
       FETCH  c1 INTO id, name1;
      select id,name1;
    -- check if the exit_loop flag has been set by mysql, close the cursor and exit the loop if it
has.
 IF exit_loop THEN
        CLOSE c1;
        LEAVE L1;
    END IF;
   END LOOP L1;
 END
```

To run above Cursor using procedure give command as:

```
MySQL> call cursor_proc2()
```

Assignment to be performed:

- 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.

Group A: Assignment No : 8

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

Assignment Name: -.

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.

Theory: -

A trigger in MySQL is a set of SQL statements that reside in a system catalog. **It is a special type of stored procedure that is invoked automatically in response to an event**. Each trigger is associated with a table, which is activated on any DML statement such as **INSERT, UPDATE**, or **DELETE**.

A trigger is called a special procedure because it cannot be called directly like a stored procedure. The main difference between the trigger and procedure is that a trigger is called automatically when a data modification event is made against a table. In contrast, a stored procedure must be called explicitly.

Generally, **triggers are of two types** according to the SQL standard: row-level triggers and statement-level triggers.

Row-Level Trigger: It is a trigger, which is activated for each row by a triggering statement such as insert, update, or delete. For example, if a table has inserted, updated, or deleted multiple rows, the row trigger is fired automatically for each row affected by the insert, update, or delete statement.

Statement-Level Trigger: It is a trigger, which is fired once for each event that occurs on a table regardless of how many rows are inserted, updated, or deleted.

Why we need/use triggers in MySQL?

We need/use triggers in MySQL due to the following features:

- Triggers help us to enforce business rules.
- Triggers help us to validate data even before they are inserted or updated.
- Triggers help us to keep a log of records like maintaining audit trails in tables.
- SQL triggers provide an alternative way to check the integrity of data.
- Triggers provide an alternative way to run the scheduled task.
- Triggers increases the performance of SQL queries because it does not need to compile each time the query is executed.
- Triggers reduce the client-side code that saves time and effort.
- Triggers help us to scale our application across different platforms.
- Triggers are easy to maintain.

Limitations of Using Triggers in MySQL

- MySQL triggers do not allow to use of all validations; they only provide extended validations. **For example**, we can use the NOT NULL, UNIQUE, CHECK and FOREIGN KEY constraints for simple validations.
- Triggers are invoked and executed invisibly from the client application. Therefore, it isn't easy to troubleshoot what happens in the database layer.
- Triggers may increase the overhead of the database server.

Types of Triggers in MySQL?

We can define the maximum six types of actions or events in the form of triggers:

1. Before Insert**:** It is activated before the insertion of data into the table.
2. After Insert**:** It is activated after the insertion of data into the table.
3. Before Update**:** It is activated before the update of data in the table.
4. After Update**:** It is activated after the update of the data in the table.
5. Before Delete**:** It is activated before the data is removed from the table.
6. After Delete**:** It is activated after the deletion of data from the table.

When we use a statement that does not use INSERT, UPDATE or DELETE query to change the data in a table, the triggers associated with the trigger will not be invoked.

Trigger Syntax

How to create triggers in MySQL?

We can use the **CREATE TRIGGER** statement for creating a new trigger in MySQL. Below is the syntax of creating a trigger in MySQL:

```
CREATE TRIGGER trigger_name
  (AFTER | BEFORE) (INSERT | UPDATE | DELETE)
     ON table_name FOR EACH ROW
     BEGIN
    --variable declarations
    --trigger code
    END;
```

The trigger body can access the column's values, which are affected by the DML statement. The **NEW** and **OLD** modifiers are used to distinguish the column values **BEFORE** and **AFTER** the execution of the DML statement. We can use the column name with NEW and OLD modifiers as **OLD.col_name** and **NEW.col_name**. The OLD.column_name indicates the column of an existing row before the updation or deletion occurs. NEW.col_name indicates the column of a new row that will be inserted or an existing row after it is updated.

Trigger Example:

Create two tables emp and emp_update as shown below

- Create table emp(EmpNo INT, Lname VARCHAR(50), Salary int(5));
- Create table emp_audit (EmpNo INT, Lname VARCHAR(50), changedat DATETIME, action VARCHAR(50));

Insert values in only emp table as shown below

- Insert into emp values (101, ‘John’,9000), (102, ‘Pawar’,9000), (103, ‘Jagruti’,7000);

Write a trigger for before update on emp

```
MySQL>DELIMITER  //
MySQL>CREATE TRIGGER  t1
        AFTER UPDATE ON emp
         FOR EACH ROW
        BEGIN
           INSERT INTO emp_audit
           SET  EmpNo = OLD. EmpNo,
           Lname= OLD.Lname,
           changedat = NOW()
            action = 'update';
        END
         //
MySQL>DELIMITER ;
MySQL>SHOW TRIGGERS;       // To see information about triggers
```

Trigger will automatically run in background after update statement on emp

- UPDATE emp SET Lname = ‘Pratik’   WHERE EmpNo = 101;
- SELECT * FROM   emp;
- SELECT * FROM   emp_audit;

Assignment to Performed:

- 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.

Group A: Assignment No : 9

Title of Assignment: MYSQL database connectivity

Assignment Name: -. Implement MYSQL/Oracle database connectivity with front end language(Java) to implement Database navigation operations (add, delete, edit, etc.).

Theory: -

Software Required and Steps

• Eclipse

• JDK 1.6

• MySQL

• Java-MySQL Connector

Connection to database with Java

The interface for accessing relational databases from Java is *Java Database Connectivity (JDBC)*. Via JDBC you create a connection to the database, issue database queries and update as well as receive the results.

JDBC provides an interface which allows you to perform SQL operations independently of the instance of the used database. To use JDBC, you require the database specific implementation of the JDBC driver.

MySQL JDBC driver(Connector)

To connect to MySQL from Java, you have to use the JDBC driver from MySQL. The MySQL JDBC driver is called *MySQL Connector/J*. You find the latest MySQL JDBC driver under the following URL: http://dev.mysql.com/downloads/connector/j.

The download contains a JAR file which we require later.

To connect Java application with the MySQL database, we need to follow following steps.

1. **Driver class:** The driver class for the mysql database is **com.mysql.jdbc.Driver**.
2. **Connection URL:** The connection URL for the mysql database is **jdbc:mysql://localhost:3306/db1** where jdbc is the API, mysql is the database, localhost is the server name on which mysql is running, we may also use IP address, 3306 is the port number and sonoo is the database name. We may use any database, in such case, we need to replace the db1 with our database name.
3. **Username:** The default username for the mysql database is **root**.
4. **Password:** It is the password given by the user at the time of installing the mysql database.

In Eclipse perform following steps:

1. File - New – Java Project –Give Project Name – ok

2. In project Explorer window- right click on project name-newclass- give Class name-ok
3. In project Explorer window- right click on project name- Build path- Configure build path- Libraries- Add External Jar - JavaMySQL Connector
4. In MySQL first Create one Database with name db1 and one table with name stud(name,age)

Steps to be perform in Java to write Code:

1. Import packages

```
 import java.sql.*;
```

2. Load Driver

```
 Class.forName("oracle.jdbc.driver.OracleDriver");
```

3. Create connection

```
 Connection con = DriverManager.getConnection(
"jdbc:mysql://localhost:3306/DatabaseName","username","passwd ")
```

4.Creates a Statement object for sending SQL statements to the database

```
Statement stmt = con.createStatement() ;
```

5. Executing SQL Statements

executeUpdate – This Method is used for insert, update and delete queries

• Example

```
  String sql = “INSERT INTO stud VALUES (‘Ankita’,21)";
  stmt.executeUpdate(sql);
```

6. Get ResultSet (SELECT Query)

```
String sql1 = "SELECT name, age FROM stud";
//Write a select query in any string variable
ResultSet rs = stmt.executeQuery(sql1);
//executeQuery used to run the select query and store the result in ResultSet
while (rs.next()) //Iterate through ResultSet till data is found
{
String name1 = rs.getString(“name");
//Store name data into name1 variable

int age1 = rs.getInt(“age");

//Store age data into age1 variable

}
```

7. Close connection

• stmt.close();

• con.close();

Group B: Assignment No : 1

Title of Assignment: MongoDB Queries

Assignment Name: -.

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

Theory: -

What is NoSQL?

NoSQL is a non-relational DBMS, that does not require a fixed schema, avoids joins, and is easy to scale. The purpose of using a NoSQL database is for distributed data stores with humongous data storage needs. NoSQL is used for Big data and real-time web apps. For example, companies like Twitter, Facebook, Google collect terabytes of user data every single day.

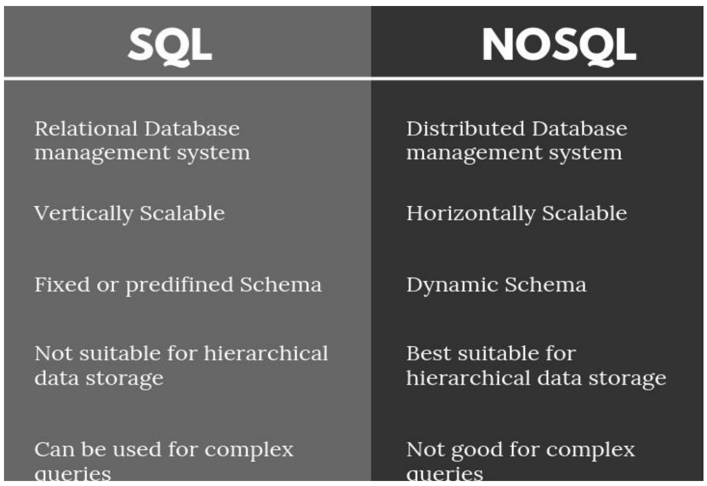
NoSQL database stands for "Not Only SQL" or "Not SQL." Though a better term would be "NoREL", NoSQL caught on. Carl Strozz introduced the NoSQL concept in 1998.

Why NoSQL?

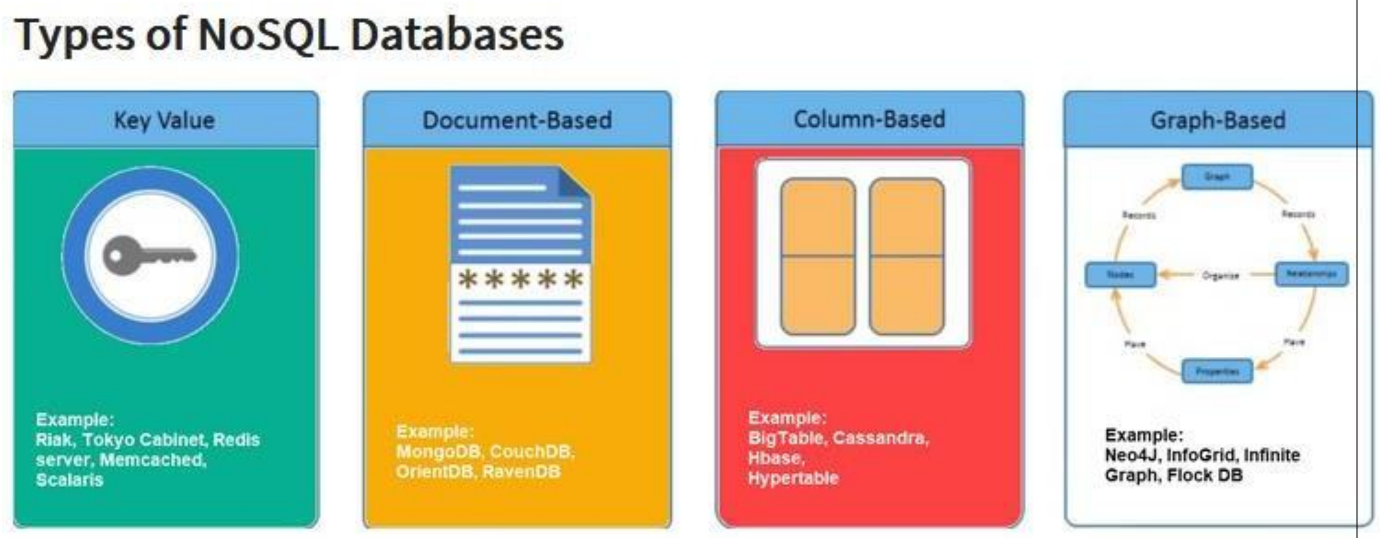
The concept of NoSQL databases became popular with Internet giants like Google, Facebook, Amazon, etc. who deal with huge volumes of data. The system response time becomes slow when you use RDBMS for massive volumes of data.

To resolve this problem, we could "scale up" our systems by upgrading our existing hardware. This process is expensive.

Difference Between SQL and NoSQL



SQL	NOSQL
Relational Database management system	Distributed Database management system
Vertically Scalable	Horizontally Scalable
Fixed or predifined Schema	Dynamic Schema
Not suitable for hierarchical data storage	Best suitable for hierarchical data storage
Can be used for complex queries	Not good for complex queries


Types of NoSQL Databases
Key Value
Example:
Riak, Tokyo Cabinet, Redis server, Memcached, Scalaris
Document-Based
Example:
MongoDB, CouchDB, OrientDB, RavenDB
Column-Based
Example:
BigTable, Cassandra, Hbase, Hypertable
Graph-Based
Graph
Records
Records
Nodes
Organize
Relationships
Have
Have
Properties
Example:
Neo4J, InfoGrid, Infinite Graph, Flock DB

Document-Oriented:
Document-Oriented NoSQL DB stores and retrieves data as a key value pair but the value part is stored as a document. The document is stored in JSON or XML formats. The value is understood by the DB and can be queried.

MongoDB

Scalable High-Performance Open-source, Document-orientated database.

• Built for Speed

• Rich Document based queries for Easy readability.

• Full Index Support for High Performance.

• Replication and Failover for High Availability.

• Auto Sharding for Easy Scalability.

• Map / Reduce for Aggregation.

Advantages of MongoDB

- Schema less : Number of fields, content and size of the document can be differ from one document to another.
- No complex joins
- Data is stored as JSON style
- Index on any attribute
- Replication and High availability

Mongo DB Terminologies for RDBMS concepts

RDBMS	MongoDB
Database	Database
Table, View	Collection
Row	Document (JSON, BSON)
Column	Field
Index	Index
Join	Embedded Document
Foreign Key	Reference
Partition	Shard

Data Types of MongoDB

- String : This is most commonly used datatype to store the data. String in mongodb must be UTF-8 valid.
- Integer : This type is used to store a numerical value. Integer can be 32 bit or 64 bit depending upon your server.
- Boolean : This type is used to store a boolean (true/ false) value.
- Double : This type is used to store floating point values.
- Min/ Max keys : This type is used to compare a value against the lowest and highest BSON elements.
- Arrays : This type is used to store arrays or list or multiple values into one key.
- Timestamp : ctimestamp. This can be handy for recording when a document has been modified or added.
- Object : This datatype is used for embedded documents.
- Null : This type is used to store a Null value.
- Symbol : This datatype is used identically to a string however, it's generally reserved for languages that use a specific symbol type.
- Date : This datatype is used to store the current date or time in UNIX time format. You can specify your own date time by creating object of Date and passing day, month, year into it.
- Object ID : This datatype is used to store the document's ID.
- Binary data : This datatype is used to store binay data.
- Code : This datatype is used to store javascript code into document.
- Regular expression : This datatype is used to store regular expression

Basic Database Operations

- use *<database name>*

switched to database provided with command

- db
  To check currently selected database use the command db
- show dbs
  Displays the list of databases
- db.dropDatabase()
  To Drop the database

- db.createCollection (name)
- Ex:- db.createCollection(Stud)
  - To create collection
- >show collections
  - List out all names of collection in current database
- db.*databasename*.insert
- ({Key : Value})
- Ex:- db.Stud.insert({{Name:"Jiya"}})
  - In mongodb you don't need to create collection. MongoDB creates collection automatically, when you insert some document.
- db.collection.drop() Example:- db.Stud.drop()

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

CRUD Operations:

- Insert
- Find
- Update
- Delete

CRUD Operations – Insert

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

Syntax

>db.COLLECTION_NAME.insert(document)

Example

>db.stud.insert({name: "Jiya", age:15})

_id Field

- If the document does not specify an *_id* field, then MongoDB will add the _id field and assign a unique *ObjectId* for the document before inserting.
- The _id value must be unique within the collection to avoid duplicate key error.

Insert a Document without Specifying an _id Field

- db.stud.insert({ Name : “Reena", Rno: 15 })
- db.stud.find()

  { "_id" : "5063114bd386d8fadbd6b004”, “Name” : “Reena", “Rno”: 15 }

Insert a Document Specifying an _id Field

- db.stud.insert({ _id: 10, Name : “Reena", Rno: 15 })
- db.stud.find()

{ "_id" : 10, “Name” : “Reena", “Rno”: 15 }

Insert Single Documents

db.stud.insert ({Name: “Ankit”, Rno:1, Address: “Pune”})

Insert Multiple Documents

```
db.stud.insert ( [
 { Name: “Ankit”, Rno:1, Address: “Pune”} ,
 { Name: “Sagar”, Rno:2},
 { Name: “Neha”, Rno:3}
 ] )
```

Insert Multicolumn attribute

```
db.stud.insert( {
    Name: “Ritu",
    Address: { City: “Pune", State: “MH” },
   Rno: 6
  })
```

Insert Multivalued attribute

```
db.stud.insert( {
     Name : “Sneha",
     Hobbies: [“Singing”, “Dancing” , “Cricket”] ,
     Rno:8
  })
```

Insert Multivalued with Multicolumn attribute

```
db.stud.insert(  {
     Name : “Sneha",
      Awards: [ { Award : “Dancing”, Rank: “1st”, Year: 2008 },
                 {Award : “Drawing”, Rank: “3rd”, Year: 2010 } ,
                 {Award : “Singing”, Rank: “1st”, Year: 2015 } ],
    Rno: 9  })
```

CRUD Operations – **Find**

The find() Method- To display data from MongoDB collection. Displays all the documents in a non structured way.

Syntax

>db.COLLECTION_NAME.find()

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

Syntax

>db. COLLECTION_NAME.find().pretty()

Specify Equality Condition

use the query document { <field>: <value> }

Examples:

- db.stud.find(name: "Jiya" })
- db.stud.find({ _id: 5 })

Comparison Operators

Operator	Description
$eq	Matches values that are equal to a specified value.
$gt	Matches values that are greater than a specified value.
$gte	values that are greater than or equal to a specified value.
$lt	Matches values that are less than a specified value.
$lte	Matches values that are less than or equal to a specified value.
$ne	Matches all values that are not equal to a specified value.
$in	Matches any of the values specified in an array.
$nin	Matches none of the values specified in an array.

Find Examples with comparison operators

- db.stud.find({ rno: { $gt:5} }) *Shows all documents whose rno>5*
- db.stud.find({ rno: { $gt: 0, $lt: 5} }) *Shows all documents whose rno greater than 0 and less than 5*

Examples to show only particular columns

- db.stud.find({name: "Jiya"},{Rno:1}) *To show the rollno of student whose name is equal to Jiya (by default _id is also shown)*
- db.stud.find({name: "jiya"},{_id:0,Rno:1}) *show the rollno of student whose name is equal to Jiya (_id is not shown)*

Examples for Sort function

- db.stud.find().sort({ Rno: 1 })
  Sort on age field in Ascending order (1)
- db.stud.find().sort({ Rno: -1 })
  Sort on age field in Ascending order(-1)

Examples of Count functions

- db.stud.find().count()
  Returns no of documents in the collection

Examples of limit and skip

- db.stud.find().limit(2)
  Returns only first 2 documents
- db.stud.find().skip(5)
  Returns all documents except first 5 documents

CRUD Operations – Update

Syntax

db.*CollectionName*.update (

```
 <query/Condition>,
 <update with $set or $unset>,
 {
   upsert: <boolean>,
   multi: <boolean>,
 } )
```

upsert

- If set to *True*, creates new document if no matches found.

multi

- If set to *True*, updates multiple documents that matches the query criteria

CRUD Operations – Update Examples

1**>** Set age = 25 where id is 100, First Whole document is replaced where condition is matched and only one field is remained as age:25

```
db.stud.update(
{ _id: 100 },
{ age: 25})
```

2> Set age = 25 where id is 100, Only the age field of one document is updated where condition is matched .

```
db.stud.update(
{ _id: 100 },
{ $set:{age: 25}})
```

3> To remove a age column from single document where id=100

```
db.stud.update(
{ _id: 100 },
{ $unset:{age: 1}})
```

CRUD Operations – Remove

- **Remove All Documents**
  - db.inventory.remove({})
- **Remove All Documents that Match a Condition**
  - db.inventory.remove ({ type : "food" })
- **Remove a Single Document that Matches a Condition**
  - db.inventory.remove ({ type : "food" }, 1)

Group B: Assignment No : 2

Title of Assignment: MongoDB Aggregation and Indexing

Assignment Name: -. Design & Develop MongoDB Queries using Aggregation and Indexing with suitable example using MongoDB

Theory: -

Indexing: Indexes support the efficient execution of queries in MongoDB

Indexing Types

- **Single field index** only includes data from a single field of the Single Field Indexes documents in a collection.
- **Compound index** includes more than one field of the documents in Compound Indexes a collection.
- **Multikey index** is an index on an array field, adding an index key for Multikey each value in the array. Indexes
- **Geospatial indexes** support location-based searches. Geospatial Indexes and Queries Text Indexes
- **Text indexes** support search of string content in documents.
- **Hashed Index** -Hashed indexes maintain entries with hashes of the values of the indexed field and are used with sharded clusters to support hashed shard keys.

Index Properties:

Index Properties -The properties you can specify when building indexes.

1. TTL Indexes The TTL index is used for TTL collections, which expire data after a period of time
2. Unique Indexes A unique index causes MongoDB to reject all documents that contain a duplicate value for the indexed field.
3. Sparse Indexes A sparse index does not index documents that do not have the indexed field.

Index Creation:

Syntax:

```
db.CollectionName.createIndex( { KeyName: 1 or -1})
```

- 1 for Ascending Sorting
- -1 for Descending Sorting

Index Creation Example:

- Single: db.stud.createIndex({ zipcode: 1})
- Compound: db.stud.createIndex({ dob: 1, zipcode: -1 })
- Unique: db.stud.createIndex({ rollno: 1 }, { unique: true })
- Sparse: db.stud.createIndex({ age: 1 }, { sparse: true })

Index Display

db.collection.getIndexes()

Returns an array that holds a list of documents that identify and describe the existing indexes on the collection.

Index Drop

Syntax:

1. db.collection.dropIndex()
2. db.collection.dropIndex(index)

Example:

1. db.stud.dropIndex()
2. db.stud.dropIndex({ "name" : 1 })

Indexing and Querying

create an ascending index on the field name for a collection records:

- db.records.createIndex({ name: 1 })
  This index can support an ascending sort on name :
- db.records.find().sort({ name: 1 })
  The index can also support descending sort
- db.records.find().sort({ a: -1 })
- db.stud.findOne({rno:2}), using index {rno:1}

Indexing with Unique:

- db.collectionname.ensureIndex ({x:1}, {unique:true})
- Don't allow {_id:10,x:2} and {_id:11,x:2}
- Don't allow {_id:12} and {_id:13} (both match {x:null}

Aggregation:

Aggregations operations process data records and return computed results.

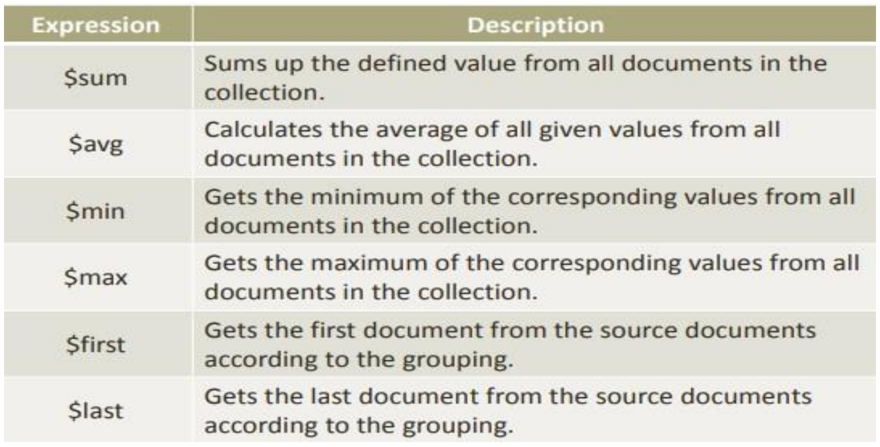
Aggregation operations group values from multiple documents together, and can perform a variety of operations on the grouped data

For aggregation in mongodb use aggregate() method.

Syntax:

- >db.COLLECTION_NAME.aggregate(AGGREGATE_OPERATION)

aggregate() method



Expression	Description
$sum	Sums up the defined value from all documents in the collection.
$avg	Calculates the average of all given values from all documents in the collection.
$min	Gets the minimum of the corresponding values from all documents in the collection.
$max	Gets the maximum of the corresponding values from all documents in the collection.
$first	Gets the first document from the source documents according to the grouping.
$last	Gets the last document from the source documents according to the grouping.

Possible stages in aggregation

- $project – Used to select some specific fields from a collection.
- $match – This is a filtering operation and thus this can reduce the amount of documents that are given as input to the next stage.
- $group – This does the actual aggregation as discussed above.
- $sort – Sorts the documents.
- $skip – With this, it is possible to skip forward in the list of documents for a given amount of documents.
- $limit – This limits the amount of documents to look at, by the given number starting from the current positions.
- $unwind – This is used to unwind document that are using arrays. When using an array, the data is kind of pre-joined and this operation will be undone with this to have individual documents again. Thus with this stage we will increase the amount of documents for the next stage.

Group B: Assignment No : 3

Assignment Name: -. MongoDB Map Reduce

Title of Assignment: Implement Map reduce operation with suitable example using MongoDB.

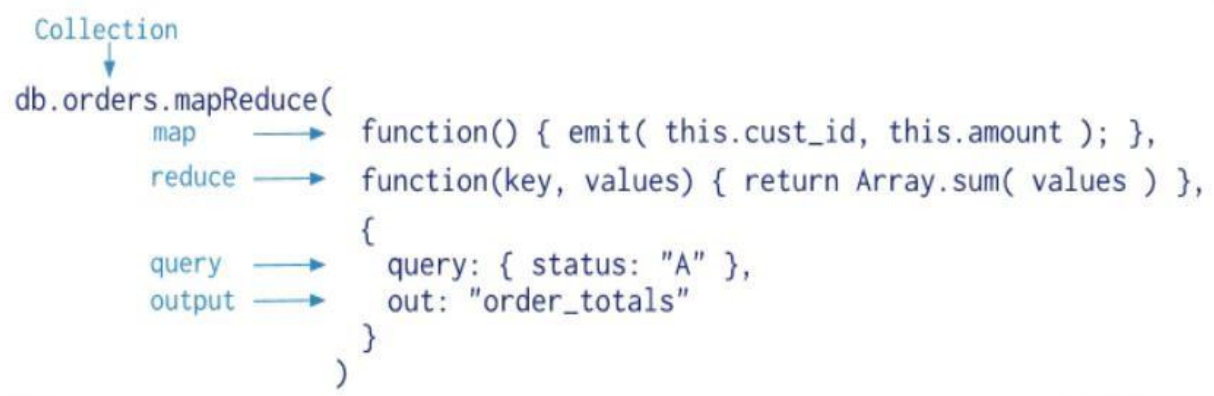
Theory: -

Map-Reduce

- **Map-reduce is a data processing paradigm for condensing**

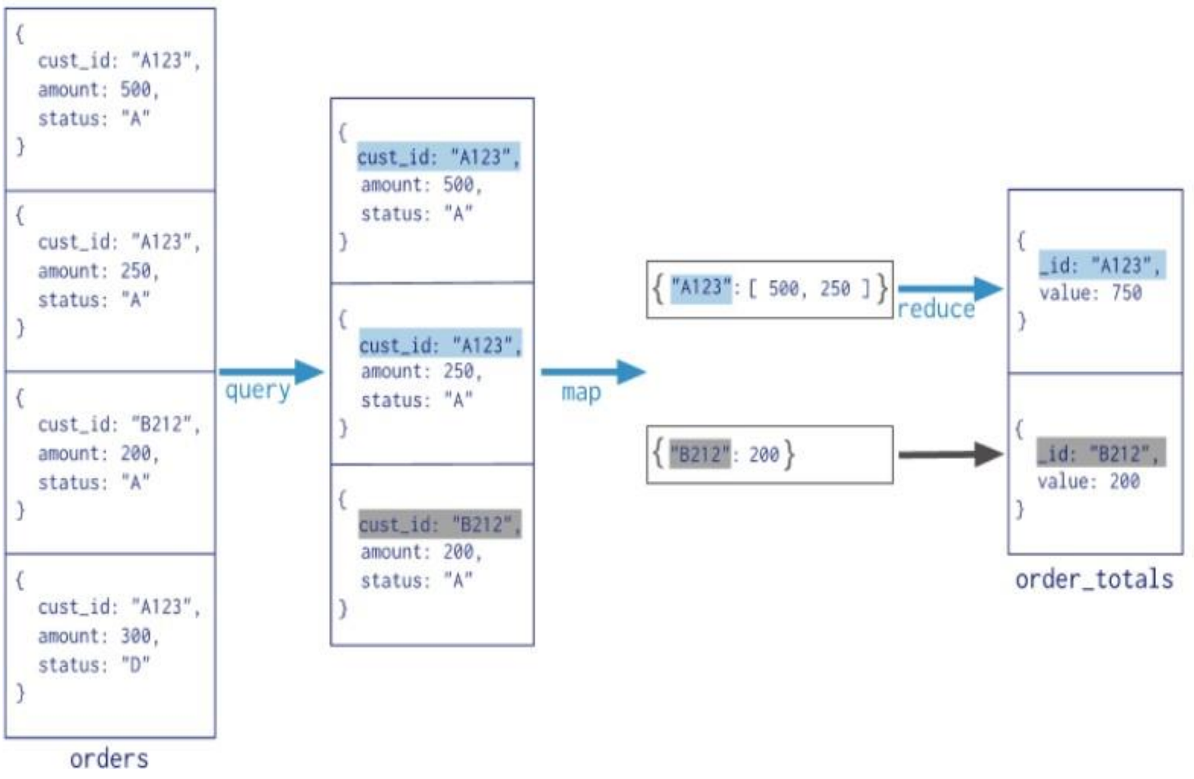
large volumes of data into useful aggregated results. For mapreduce operations, MongoDB provides the mapReduce database command.

- **Consider the following map-reduce operation:**



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

MapReduce:



```
{
  cust_id: "A123",
  amount: 500,
  status: "A"
}
{
  cust_id: "A123",
  amount: 250,
  status: "A"
}
{
  cust_id: "B212",
  amount: 200,
  status: "A"
}
{
  cust_id: "A123",
  amount: 300,
  status: "D"
}
```

```
{
  cust_id: "A123",
  amount: 500,
  status: "A"
}
{
  cust_id: "A123",
  amount: 250,
  status: "A"
}
{
  cust_id: "B212",
  amount: 200,
  status: "A"
}
```

```
{ "A123": [ 500, 250 ] }
{ "B212": 200 }
```

```
{
  _id: "A123",
  value: 750
}
{
  _id: "B212",
  value: 200
}
```

Map-Reduce:

In very simple terms, the mapReduce command takes 2 primary inputs, the mapper function and the reducer function.

A Mapper will start off by reading a collection of data and building a Map with only the required fields we wish to process and group them into one array based on the key.

And then this key value pair is fed into a Reducer, which will process the values.

Map-Reduce Syntax:

```
db.collection.mapReduce(
 function() {emit(key, value);},
function(key,values) {return reduceFunction},
 {
 out: collection,
 query: document,
 sort: document,
 limit: number
 }
)
```

Map-Reduce Syntax Explanation:

The above map-reduce function will query the collection, and then map the output documents to the emit key-value pairs. After this, it is reduced based on the keys that have multiple values. Here, we have used the following functions and parameters.

• Map: – It is a JavaScript function. It is used to map a value with a key and produces a key-value pair.

• Reduce: – It is a JavaScript function. It is used to reduce or group together all the documents which have the same key.

• Out: – It is used to specify the location of the map-reduce query output.

• Query: – It is used to specify the optional selection criteria for selecting documents.

• Sort: – It is used to specify the optional sort criteria.

• Limit: – It is used to specify the optional maximum number of documents which are desired to be returned.

Group B: Assignment No : 4

Title of Assignment: Java & Mongo DB database connectivity.

Assignment Name: -. Write a program to implement Mongo DB database connectivity with Front End Language(Java) Implement Database navigation operations (add, delete, edit etc.).

Theory: -

Software Required and Steps

- Eclipse
- JDK 1.6
- MongoDB
- MongoDB-Java-Driver

In Eclipse perform following steps:

1. File - New – Java Project –Give Project Name – ok

2. In project Explorer window- right click on project namenew- class- give Class name- ok

3. In project Explorer window- right click on project nameBuild path- Configure build path- Libraries- Add External

Jar - MongoDB-Java-Driver

4. Start Mongo server before running the program

Steps to Write Code in Java

1. **Import packages**

```
import com.mongodb.*;
```

2. Create connection

```
MongoClient mongo = new MongoClient( "localhost" , 27017 );
```

3. Create Database

```
DB db = mongo.getDB("database name");
```

4. Insert Document

```
BasicDBObject d1 = new BasicDBObject(“rno“,“1”).append(“name", “Monika"). append(“age", “17”)
coll.insert(d1);
```

5. Display document

```
DBCursor cursor = coll.find();
while (cursor.hasNext())
 {
 System.out.println(cursor.next());
 }
```

6. Update Document

- BasicDBObject query = new BasicDBObject();
- query.put("name", “Monika");
- BasicDBObject newDocument = new BasicDBObject();
- newDocument.put("name", “Ragini");
- BasicDBObject updateObj = new BasicDBObject();
- updateObj.put("$set", newDocument);
- Coll.update(query, updateObj);

7. Remove document

```
BasicDBObject searchQuery = new BasicDBObject("name", “Monika");
Coll.remove(searchQuery);
```