ASSIGNMENT NUMBER: A1

Revised On: 15/06/2018

TITLE	Study of Open source relational database: MySQL
PROBLEM STATEMENT /DEFINITION	To study open source relational MySql.
OBJECTIVE	To learn and understand the basic database architecture and the various components of it.
S/W PACKAGES AND HARDWARE APPARATUS USED	MySQL PC with the configuration as Latest Version of 64 bit Operating Systems, Open Source Fedora-GHz. 8 G.B. RAM, 500 G.B. HDD, 15"Color Monitor, Keyboard, Mouse
REFERENCES	Silberschatz A., Korth H., Sudarshan S., "Database System Concepts", 5th Edition, McGraw Hill Publishers, 2002, ISBN 0-07-120413-X Connally T., Begg C., "Database Systems", 3rd Edition, Pearson Education, 2002, ISBN 81-7808-861-4 mysql.com/docs/mysql-tutorial-excerpt-5.1-en.pdf
STEPS	Refer to details
INSTRUCTIONS FOR WRITING	• Date • Title • Problem Definition

JOURNAL	• Learning Objective • Learning Outcome • Theory-Related concept,Architecture,Syntax etc • Class Diagram/ER diagram • Test cases • Program Listing • Output • Conclusion

Aim: Study of Open source relational database: MySQL.

Pre-requisite:

1. Basics of File handling, Databases

Learning Objectives:

- To learn and understand the basic database architecture and the various components of it.

Learning Outcomes:

The students will be able to

- Understand the basic database architecture and the various components of it.

Theory:

MySQL is a fast, easy-to-use RDBMS being used for many small and big businesses. MySQL is developed, marketed, and supported by MySQL AB, which is a Swedish company. MySQL is becoming so popular because of many good reasons:

MySQL is released under an open-source license. So you have nothing to pay to use it. 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. MySQL is very friendly to PHP, the most appreciated language for web development. MySQL supports large databases, up to 50 million rows or more in a table. The default file size limit for a table is 4GB, but you can increase this (if your operating system can handle it) to a theoretical limit of 8 million terabytes (TB). MySQL is customizable. The open-source GPL license allows programmers to modify the MySQL software to fit their own specific environments.

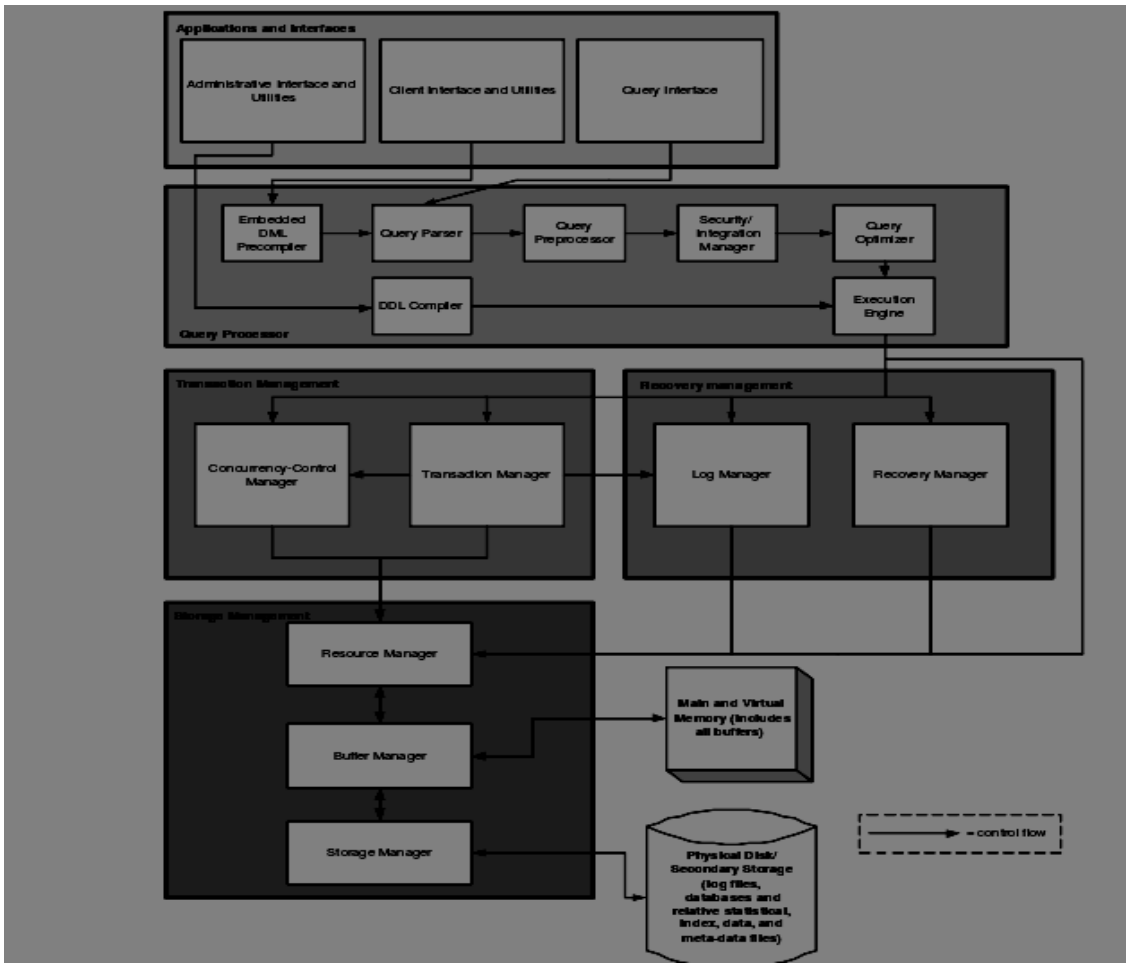


Fig 1.Database Architecture

1. **Application Layer and Interfaces**

MySQL application layer is where the clients and users interact with the MySQL RDBMS. There are three components in this layer as can be seen in the layered. These components illustrate the different kinds of users that can interact with the MySQL RDBMS, which are the administrators, clients and query users. The administrators use the administrative interface and utilities. In MySQL, some of these utilities are mysqladmin which performs tasks like shutting

down the server and creating or dropping databases. Clients communicate to the MySQL RDBMS through the interface or utilities. The client interface uses MySQL APIs for various different programming languages such as the C API, DBI API for Perl, PHP API, Java API, Python API, MySQL C++ API and Tcl. Query users interact with the MySQL RDBMS through a query interface that is mysql. Mysql is a monitor (interactive program) that allows the query users to issue SQL statements to the server and view the results.

2. Logical Layer

It was found that MySQL does indeed have a logical architecture . The MySQL documentation gave an indication as to precisely how these modules could be further broken down into subsystems arranged in a layered hierarchy corresponding to the layered architecture in Garlan and Shaw. The following section details these subsystems and the interactions within them.

2.1 Query Processor

The vast majority of interactions in the system occur when a user wishes to view or manipulate the underlying data in storage. These queries, which are specified using a data-manipulation language (ie SQL), are parsed and optimized by a query processor. This processor, depicted in Figure 3 above, can be represented as pipeline and filter architecture in the sense of Garlan and Shaw where the result of the previous component becomes an input or requirement to the next component. The component architecture of the query processor will be explained below.

2.1.1 Embedded DML Precompiler

When a request is received from a client in the application layer, it is the responsibility of the embedded DML (Data Manipulation Language) precompiler to extract the relevant SQL statements embedded in the client API commands, or to translate the client commands into the corresponding SQL statements. This is the first step in the actual processing of a client application written in a programming language such as C++ or Perl, before compiling the SQL query. The client request could come from commands executed from an application interface (API), or an application program. This is prevalent in all general RDBMS's. MySQL has this component in order to process the MySQL client application request into the format that MySQL understands.

2.1.2 DDL Compiler

Requests to access the MySQL databases received from an administrator are processed by the DDL (Data Definition Language) compiler. The DDL compiler compiles the commands (which are SQL statements) to interact directly with the database. The administrator and administrative utilities do not expose an interface, and hence execute directly to the MySQL server. Therefore, the embedded DML precompiler does not process it, and this explains the need for a DDL compiler.

2.1.3 Query Parser

After the relevant SQL query statements are obtained from deciphering the client request or the administrative request, the next step involves parsing the MySQL query. In this stage, the objective of the query parser is to create a parse tree structure based on the query so that it can be easily understood by the other components later in the pipeline.

2.1.4 Query Preprocessor

The query parse tree, as obtained from the query parser, is then used by the query preprocessor to check the SQL syntax and check the semantics of the MySQL query to determine if the query is valid. If it is a valid query, then the query progresses down the pipeline. If not, then the query does not proceed and the client is notified of the query processing error.

2.1.5 Security/Integration Manager

Once the MySQL query is deemed to be valid, the MySQL server needs to check the access control list for the client. This is the role of the security integration manager which checks to see if the client has access to connecting to that particular MySQL database and whether he/she has table and record privileges. In this case, this prevents malicious users from accessing particular tables and records in the database and causing havoc in the process.

2.1.6 Query Optimizer

After determining that the client has the proper permissions to access the specific table in the database, the query is then subjected to optimization. MySQL uses the query optimizer for

executing SQL queries as fast as possible. As a result, this is the reason why the performance of MySQL is fast compared to other RDBMS's. The task of the MySQL query optimizer is to analyze the processed query to see if it can take advantage of any optimizations that will allow it to process the query more quickly. MySQL query optimizer uses indexes whenever possible and uses the most restrictive index in order to first eliminate as many rows as possible as soon as possible. Queries can be processed more quickly if the most restrictive test can be done first.

2.1.7 Execution Engine

Once the MySQL query optimizer has optimized the MySQL query, the query can then be executed against the database. This is performed by the query execution engine, which thenp roceeds to execute the SQL statements and access the physical layer of the MySQL database . As well the database administrator can execute commands on the database to perform specific tasks such as repair, recovery, copying and backup, which it receives from the DDL compiler.

2.1.8 Scalability/Evolvability

The layered architecture of the logical layer of the MySQL RDBMS supports the evolvability of the system. If the underlying pipeline of the query processor changes, the other layers in the RDBMS are not affected. This is because the architecture has minimal sub-component interactions to the layers above and below it, as can be seen from the architecture diagram. The only sub-components in the query processor that interact with other layers is the embedded DML preprocessor, DDL compiler and query parser (which are at the beginning stages of the pipeline) and the execution engine (end of the pipeline). Hence, if the query preprocessor security/integration manager and/or query optimizer is replaced, this does not affect the outcome of the query processor.

2.2.1 Transaction Manager

As of version MySQL 4.0.x, support was added for transactions in MySQL. A transaction is a single unit of work that has one or more MySQL commands in it. The transaction manager is responsible for making sure that the transaction is logged and executed atomically. It does so through the aid of the log manager and the concurrency-control manager. Moreover, the transaction manager is also responsible for resolving any deadlock situations that occur. This

situation can occur when two transactions cannot continue because they each have some data that the other needs to proceed. Furthermore, the transaction manager is responsible for issuing the COMMIT and the ROLLBACK SQL commands.The COMMIT command commits to performing a transaction. Thus, a transaction is incomplete until it is committed to. The ROLLBACK command is used when a crash occurs during the execution of a transaction. If a transaction were left incomplete, the ROLLBACK command would undo all changes made by that transaction. The result of executing this command is restoring the database to its last stable state.

2.2.2 Concurrency-Control Manager

The concurrency-control manager is responsible for making sure that transactions are executed separately and independently. It does so by acquiring locks, from the locking table that is stored in memory, on appropriate pieces of data in the database from the resource manager. Once the lock is acquired, only the operations in one transaction can manipulate the data. If a different transaction tries to manipulate the same locked data, the concurrency-control manager rejects the request until the first transaction is complete.

2.3 Recovery Management

2.3.1 Log Manager

The log manager is responsible for logging every operation executed in the database. It does so by storing the log on disk through the buffer manager. The operations in the log are stored as MySQL commands. Thus, in the case of a system crash, executing every command in the log will bring back the database to its last stable state.

2.3.2 Recovery Manager

The recovery manager is responsible for restoring the database to its last stable state. It does so by using the log for the database, which is acquired from the buffer manager, and executing each operation in the log. Since the log manager logs all operations performed on the database (from the beginning of the database’s life), executing each command in the log file would recover the database to its last stable state.

2.4 Storage Management

Storage is physically done on some type of secondary storage, however dynamic access of this medium is not practical. Thus, all work is done through a number of buffers. The buffers reside in main and virtual memory and are managed by a Buffer Manager. This manager works in conjunction with two other manager entities related to storage: the Resource Manager and the StorageManager.

2.4.1 Storage Manager

At the lowest level exists the Storage Manager. The role of the Storage Manager is to mediate requests between the Buffer Manager and secondary storage. The Storage Manager makes requests through the underlying disk controller (and sometimes the operating system) to retrieve data from the physical disk and reports them back to the Buffer Manager.

2.4.2 Buffer Manager

The role of the Buffer Manager is to allocate memory resources for the use of viewing and manipulating data. The Buffer Manager takes in formatted requests and decides how much memory to allocate per buffer and how many buffers to allocate per request. All requests are made from the Resource Manager.

2.4.3 Resource Manager

The purpose of the Resource Manager is to accept requests from the execution engine, put them into table requests, and request the tables from the Buffer Manager. The Resource Manager receives references to data within memory from the Buffer Manager and returns this data to the upper layers.

2.5 Evolvability/Scalability

The goals of the Transaction Management subsystem and the Recovery Management subsystem seem to provide non-functional requirements such evolvability and scalability. For example, the different managers provide the necessary abstractions so that the implementation can change while leaving the interface the same, thereby ensuring that the system can evolve to contain

better data structures or algorithms. Furthermore, these subsystems provide scalability by being able to handle several transactions from several different users concurrently, or by recovering crashes from several different databases without much effort.

FAQ questions :

1. List the different components of database architecture?
2. Explain the levels of data abstraction?
3. Explain 2-tier architecture?

Review questions:

1. Explain the DDL interpreter and DML compiler?
2. What is physical and logical independence for database?

ASSIGNMENT NUMBER: A2

Revised On: 15/06/2018

TITLE	Design and Develop SQL DDL statements which demonstrate the use of SQL objects such as creation of: Table, View, Index, Sequence, Synonym.
PROBLEM STATEMENT /DEFINITION	Implement DDL commands in context of view, index, sequence.
OBJECTIVE	• To understand & implement the various DDL Commands. • To understand database concepts like view, index ,sequence and synonym.
S/W PACKAGES AND HARDWARE APPARATUS USED	MySQL PC with the configuration as Latest Version of 64 bit Operating Systems,Open Source Fedora-GHz. 8 G.B. RAM, 500 G.B. HDD, 15"Color Monitor, Keyboard, Mouse
REFERENCES	Silberschatz A., Korth H., Sudarshan S., "Database System Concepts", 5th Edition, McGraw Hill Publishers, 2002, ISBN 0-07-120413-X Connally T., Begg C., "Database Systems", 3rd Edition, Pearson Education, 2002, ISBN 81-7808-861-4 mysql.com/docs/mysql-tutorial-excerpt-5.1-en.pdf
STEPS	Refer to details
INSTRUCTIONS FOR	• Title • Problem Definition

WRITING JOURNAL	• Learning Objective • Learning Outcome • Theory-Related concept,Architecture,Syntax etc • Class Diagram/ER diagram • Test cases • Program Listing • Output • Conclusion

Aim: Design and develop SQL DDL statements which demonstrate the use of SQL objects such as Table, View, Index ,sequence.

Pre-requisite:

- Basics of Database
- SQL DDL commands and their syntax,
- Constraints in SQL

Learning Objectives:

- To understand & implement the various DDL Commands.
- To understand database concepts like view, index ,sequence and synonym.

Learning Outcomes:

The students will be able to

- Use and Implement the DDL commands like Create the tables, Alter the table structure.

- Implement view, index (types of index),synonym and sequence concept**.**

Theory:

DDL COMMANDS: DDL is short form of Data Definition Language, which deals with data schemas and description, of how data can reside in database

Various commands in DDL are:

1. **Create :** Create table command defines each attribute uniquely. Each attribute has 3 mandatory things.

   (I) Attribute name
   (II) Attribute size
   (III) Data type

   Syntax:
   Create table tablename (Attribute_name attribute_datatype(size),Attribute_name attribute_datatype(size),.....n);

2. **Alter :**
   By using ALTER command existing table can be modified.

   Adding New Columns

   Syntax:
   ALTER TABLE <table_name> ADD (<NewColumnName> <Data_Type>(<size>),.....n);

   Dropping a Column from the Table

   Syntax :
   ALTER TABLE <table_name> DROP COLUMN <column_name>;
   This command will drop particular column.

   Modifying Existing Table

   Syntax:

ALTER TABLE <table_name> MODIFY (<column_name> <NewDataType>(<NewSize>));

Restriction on the ALTER TABLE

Using the ALTER TABLE clause the following tasks cannot be performed.

1. Change the name of the table.
2. Change the name of the column.
3. Decrease the size of a column if table data exists.

3. Drop :

The Drop command will destroy table along with the data enteries in it.

Syntax: Drop table <Tablename>;

4. Truncate :

The truncate command deletes all entries existing in tables but keep the structure of table as described.

Syntax: Truncate Table <tablename>;

5. Rename:

The rename command is used to rename the table

Syntax: Rename <OldTableName> <NewTableName>;

Creating Views:

Database views are created using the CREATE VIEW statement. Views can be created from a single table, multiple tables, or another view. To create a view, a user must have the appropriate system privilege according to the specific implementation.

The basic CREATE VIEW syntax is as follows:

CREATE VIEW view_name AS SELECT column1, column2.....FROM table_name WHERE [condition];

You can include multiple tables in your SELECT statement in very similar way as you use them in normal SQL SELECT query.

Example:

Consider the CUSTOMERS table having the following records:

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

Now, following is the example to create a view from CUSTOMERS table. This view would be used to have customer name and age from CUSTOMERS table:

SQL > CREATE VIEW CUSTOMERS_VIEW AS SELECT name, age FROM CUSTOMERS;

Now, you can query CUSTOMERS_VIEW in similar way as you query an actual table. Following is the example:

SQL > SELECT * FROM CUSTOMERS_VIEW;

This would produce the following result:

```
+----------+-----+
| name     | age |
+----------+-----+
| Ramesh   |  32 |
| Khilan   |  25 |
| kaushik  |  23 |
| Chaitali |  25 |
| Hardik   |  27 |
| Komal    |  22 |
| Muffy    |  24 |
+----------+-----+
```

The WITH CHECK OPTION:

The WITH CHECK OPTION is a CREATE VIEW statement option. The purpose of the WITH CHECK OPTION is to ensure that all UPDATE and INSERTs satisfy the condition(s) in the view definition.

If they do not satisfy the condition(s), the UPDATE or INSERT returns an error.

The following is an example of creating same view CUSTOMERS_VIEW with the WITH CHECK OPTION:

```
CREATE VIEW CUSTOMERS_VIEW AS SELECT name, age FROM  CUSTOMERS WHERE age IS NOT NULL WITH CHECK OPTION;
```

The WITH CHECK OPTION in this case should deny the entry of any NULL values in the view's AGE column, because the view is defined by data that does not have a NULL value in the AGE column.

Updating a View:

A view can be updated under certain conditions:

The SELECT clause may not contain the keyword DISTINCT.

The SELECT clause may not contain summary functions.

The SELECT clause may not contain set functions.

The SELECT clause may not contain set operators.

The SELECT clause may not contain an ORDER BY clause.

The FROM clause may not contain multiple tables.

The WHERE clause may not contain subqueries.

The query may not contain GROUP BY or HAVING.

Calculated columns may not be updated.

All NOT NULL columns from the base table must be included in the view in order for the INSERT query to function.

So if a view satisfies all the above-mentioned rules then you can update a view. Following is an example to update the age of Ramesh:

SQL > UPDATE CUSTOMERS_VIEW SET AGE = 35 WHERE name='Ramesh';

This would ultimately update the base table CUSTOMERS and same would reflect in the view itself. Now, try to query base table, and SELECT statement would produce the following result:

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 35 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

Inserting Rows into a View:

Rows of data can be inserted into a view. The same rules that apply to the UPDATE command also apply to the INSERT command.

Here we can not insert rows in CUSTOMERS_VIEW because we have not included all the NOT NULL columns in this view, otherwise you can insert rows in a view in similar way as you insert them in a table.

Deleting Rows into a View:

Rows of data can be deleted from a view. The same rules that apply to the UPDATE and INSERT commands apply to the DELETE command.

Following is an example to delete a record having AGE= 22.

SQL > DELETE FROM CUSTOMERS_VIEW WHERE age = 22;

This would ultimately delete a row from the base table CUSTOMERS and same would reflect in the view itself. Now, try to query base table, and SELECT statement would produce the following result:

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 35 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

Dropping Views:

Obviously, where you have a view, you need a way to drop the view if it is no longer needed. The syntax is very simple as given below:

DROP VIEW view_name;

Following is an example to drop CUSTOMERS_VIEW from CUSTOMERS table

DROP VIEW CUSTOMERS_VIEW;

INDEX

An index can be created in a table to find data more quickly and efficiently.

The users cannot see the indexes, they are just used to speed up searches/queries.

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

SQL CREATE INDEX Syntax

Creates an index on a table. Duplicate values are allowed:

CREATE INDEX index_name
ON table_name (column_name)

SQL CREATE UNIQUE INDEX Syntax

Creates a unique index on a table. Duplicate values are not allowed:

CREATE UNIQUE INDEX index_name
ON table_name (column_name)

CREATE INDEX Example

The SQL statement below creates an index named "PIndex" on the "LastName" column in the "Persons" table:

CREATE INDEX PIndex
ON Persons (LastName)

If you want to create an index on a combination of columns, you can list the column names within the parentheses, separated by commas:

CREATE INDEX PIndex
ON Persons (LastName, FirstName)

Creating Sequence

Syntax: CREATE Sequence sequence-name start with initial-value increment by increment-value maxvalue maximum-value cycle|nocycle

TWO-TIER AND THREE-TIER PROCESSING MODELS

The JDBC API supports both two-tier and three-tier processing models for database access.

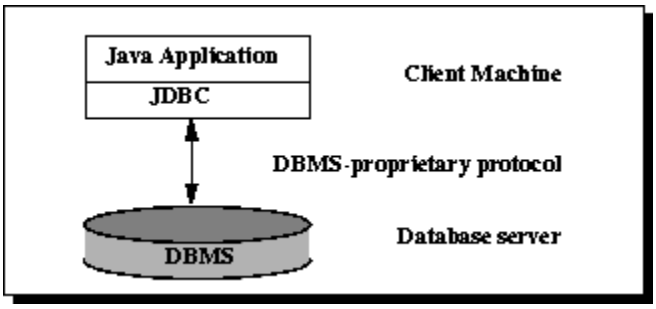


Figure 1: Two-tier Architecture for Data Access.

In the two-tier model, a Java applet or application talks directly to the data source. This requires a JDBC driver that can communicate with the particular data source being accessed. A user's commands are delivered to the database or other data source, and the results of those statements are sent back to the user. The data source may be located on another machine to which the user is connected via a network. This is referred to as a client/server configuration, with the user's machine as the client, and the machine housing the data source as the server. The network can be an intranet, which, for example, connects employees within a corporation, or it can be the Internet.

In the three-tier model, commands are sent to a "middle tier" of services, which then sends the commands to the data source. The data source processes the commands and sends the results back to the middle tier, which then sends them to the user. MIS directors find the three-tier model very attractive because the middle tier makes it possible to maintain control over access and the kinds of updates that can be made to corporate data. Another advantage is that it simplifies the deployment of applications. Finally, in many cases, the three-tier architecture can provide performance advantages.

JDBC :-

The fundamental steps involved in the process of connecting to a database and executing a query consist of the following**:**

- Import JDBC packages.
- Load and register the JDBC driver.
- Open a connection to the database.
- Create a statement object to perform a query.
- Execute the statement object and return a query resultset.
- Process the resultset.
- Close the resultset and statement objects.
- Close the connection.

These steps are described in detail in the sections that follow.

Import JDBC Packages

This is for making the JDBC API classes immediately available to the application program. The following import statement should be included in the program irrespective of the JDBC driver being used:

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

Additionally, depending on the features being used, Oracle-supplied JDBC packages might need to be imported. For example, the following packages might need to be imported while using the Oracle extensions to JDBC such as using advanced data types such as BLOB, and so on.

```
import oracle.jdbc.driver.*;
import java.sql.*;
```

Load and Register the JDBC Driver

This is for establishing a communication between the JDBC program and the Oracle database. This is done by using the static `registerDriver()` method of the `DriverManager` class of the JDBC API. The following line of code does this job:

```
String url = "jdbc:mysql://mysqlserverip:3306/test";
String userName = "abc";
 String password = "abc123";

DriverManager.getConnection(url, userName, password);
```

JDBC Driver Registration

For the entire Java application, the JDBC driver is registered only once per each database that needs to be accessed. This is true even when there are multiple database connections to the same data server.

Alternatively, the `forName()` method of the `java.lang.Class` class can be used to load and register the JDBC driver:

```
String driver = "com.mysql.jdbc.Driver";
Class.forName(driver).newInstance();
```

However, the `forName()` method is valid for only JDK-compliant Java Virtual Machines and implicitly creates an instance of the Oracle driver, whereas the `registerDriver()` method does this explicitly.

Connecting to a Database

Once the required packages have been imported and the Oracle JDBC driver has been loaded and registered, a database connection must be established. This is done by using the `getConnection()` method of the `DriverManager` class. A call to this method creates an object instance of the `java.sql.Connection` class. The `getConnection()` requires three input parameters, namely, a connect string, a username, and a password. The connect string should specify the JDBC driver to be yes and the database instance to connect to.

The `getConnection()` method is an overloaded method that takes

- Three parameters, one each for the URL, username, and password.
- Only one parameter for the database URL. In this case, the URL contains the username and password.

The following lines of code illustrate using the `getConnection()` method:

Connection conn = DriverManager.getConnection(URL, username, passwd);
Connection conn = DriverManager.getConnection(URL);

where URL, username, and passwd are of `String` data types.

Querying the Database

Querying the database involves two steps: first, creating a statement object to perform a query, and second, executing the query and returning a resultset.

Creating a Statement Object

This is to instantiate objects that run the query against the database connected to. This is done by the `createStatement()` method of the `conn Connection` object created above. A call to this method creates an object instance of the `Statement` class. The following line of code illustrates this:

Statement sql_stmt = conn.createStatement();

Executing the Query and Returning a ResultSet

Once a `Statement` object has been constructed, the next step is to execute the query. This is done by using the `executeQuery()` method of the `Statement` object. A call to this method takes as parameter a SQL SELECT statement and returns a JDBC `ResultSet` object. The following line of code illustrates this using the `sql_stmt` object created above:

```
ResultSet rset = sql_stmt.executeQuery
    ("SELECT empno, ename, sal, deptno FROM emp ORDER BY ename");
```

Alternatively, the SQL statement can be placed in a string and then this string passed to the executeQuery() function. This is shown below.

```
String sql = "SELECT empno, ename, sal, deptno FROM emp ORDER BY ename";
ResultSet res = sql_stmt.executeQuery(sql);
```

Statement and ResultSet Objects

Statement and ResultSet objects open a corresponding cursor in the database for SELECT and other DML statements.

The above statement executes the SELECT statement specified in between the double quotes and stores the resulting rows in an instance of the `ResultSet` object named `rset`.

Processing the Results of a Database Query That Returns Multiple Rows

Once the query has been executed, there are two steps to be carried out:

- Processing the output resultset to fetch the rows
- Retrieving the column values of the current row

The first step is done using the `next()` method of the `ResultSet` object. A call to `next()` is executed in a loop to fetch the rows one row at a time, with each call to `next()` advancing the control to the next available row. The `next()` method returns the Boolean value true while rows are still available for fetching and returns false when all the rows have been fetched.

The second step is done by using the `getXXX()` methods of the JDBC rset object. Here `getXXX()` corresponds to the `getInt()`, `getString()` etc with XXX being replaced by a Java datatype.

The following code demonstrates the above steps:

```
while (res.next()) {
        int id = res.getInt("id");
        String msg = res.getString("ename");
        System.out.println(id + "\t" + ename);
        }
```

Specifying `get()` Parameters

The parameters for the `getXXX()` methods can be specified by position of the corresponding columns as numbers 1, 2, and so on, or by directly specifying the column names enclosed in double quotes, as `getString("ename")` and so on, or a combination of both.

Closing the ResultSet and Statement

Once the `ResultSet` and `Statement` objects have been used, they must be closed explicitly. This is done by calls to the `close()` method of the `ResultSet` and `Statement` classes. The following code illustrates this:

```
res.close();
sql_stmt.close();
```

If not closed explicitly, there are two disadvantages:

- Memory leaks can occur
- Maximum Open cursors can be exceeded

Closing the `ResultSet` and `Statement` objects frees the corresponding cursor in the database.

Closing the Connection

The last step is to close the database connection opened in the beginning after importing the packages and loading the JDBC drivers. This is done by a call to the `close()` method of the Connection class.

The following line of code does this:

```
conn.close();
```

FAQs :

1. How to establish referential identity?
2. Which referential actions to be used while creating tables?
3. What are DDL commands?
4. Explain sequence and synonyms?
5. What is view. Types of view?
6. What is index. Explain types of index?
7. What is JDBC and JDBC Driver?
8. What are the steps to connect to database in Java?
9. What is difference between Statement and PreparedStatement interface?
10. Which interface is responsible for transaction management in JDBC?

Oral/Review Questions:

1. What are DDL commands?
2. Explain sequence and synonyms?
3. What is view. Types of view?
4. What is index. Explain types of index?

ASSIGNMENT NUMBER: A3

Revised On: 15/06/2018

TITLE	Design at least 10 SQL queries for suitable database application using SQL.
PROBLEM STATEMENT /DEFINITION	Design at least 10 SQL queries for suitable database application using SQL DML statements: Insert, Select, Update, Delete with operators ,functions and set operators
OBJECTIVE	• To understand & implement the various DML Commands. • To understand database concepts like functions and set operators.
S/W PACKAGES AND HARDWARE APPARATUS USED	MY SQL PC with the configuration as Latest Version of 64 bit Operating Systems,Open Source Fedora-GHz. 8 G.B. RAM, 500 G.B. HDD, 15"Color Monitor, Keyboard, Mouse
REFERENCES	• Silberschatz A., Korth H., Sudarshan S., "Database System Concepts", 5th Edition, McGraw Hill Publishers, 2002, ISBN 0-07-120413-X • Connally T., Begg C., "Database Systems", 3rd Edition, Pearson Education, 2002, ISBN 81-7808-861-4
STEPS	Refer to details
INSTRUCTIONS FOR WRITING JOURNAL	• Title • Problem Definition • Learning Objectives • Theory

	• Class Diagram/ER Diagram • Test cases • Program Listing • Output • Conclusion

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

Pre-requisite:

- Basics of Database.
- SQL DML commands and their syntax.
- Functions and set operators

Learning Objectives:

- To understand & implement the various DML Commands.
- To understand database concepts like functions and set operators.

Learning Outcomes:

The students will be able to

1. Implement the various DML Commands with options.
2. Implement database concepts like functions and set operators.

Theory:

DML is short name of Data Manipulation Language which deals with data manipulation, and includes most common SQL statements such SELECT, INSERT, UPDATE, DELETE etc, and it is used to store, modify, retrieve, delete and update data in database.

SELECT: MySQL SELECT statement is used to fetch data from a database table.

Syntax: SELECT column_name(s) FROM table_name

INSERT: MySQL Query statement "INSERT " is used to insert new records in a table

Syntax: INSERT INTO table_name (column, column1, column2, column3, ...) VALUES (value, value1, value2, value3 ...)

UPDATE: The UPDATE statement is used to modify data in a table.

Syntax: UPDATE table_name

SET column=value, column1=value1,...

WHERE someColumn=someValue

DELETE: The DELETE FROM statement is used to delete data from a database table.

Syntax: DELETE FROM tableName

WHERE someColumn = someValue

SET-OPERATORS:-

UNION: It returns a union of two select statements. It is returning unique (distinct) values of them.

Syntax: SELECT * FROM table1
UNION
SELECT * FROM table2;

UNION ALL

Similar to UNION just that UNION ALL returns also the duplicated values.

Syntax: SELECT * FROM table1
UNION
SELECT * FROM table2;

When using UNION and UNION ALL columns in SELECT statements need to match. This would return an error:

Syntax : SELECT column1 FROM table1
UNION
SELECT * FROM table2;

MINUS

MINUS (also known as EXCEPT) returns the difference between the first and second SELECT statement. It is the one where we need to be careful which statement will be put first, cause we will get only those results that are in the first SELECT statement and not in the second.

Syntax: SELECT * FROM table1
MINUS
SELECT * FROM table2;

INTERSECT

INTERSECT is opposite from MINUS as it returns us the results that are both to be found in first and second SELECT statement.

Syntax: SELECT * FROM table1
INTERSECT
SELECT * FROM table2;

Test Cases:

Description	Expected Output	Actual Output	
1.Create table	Table created with attributed and constraints	Successful	Unsuccessful on syntax error
2.Insert	Insert new entry	Successful	Unsuccessful on syntax error
3.Alter	Alter cell of a table	Successful	Unsuccessful on syntax error
3.1 Alter Add	Add column in table	Successful	
3.2 Alter Modify	Modify Table entries	Successful	
3.3 Alter Drop	Drop attributes	Successful	
4.Select	Select specific rows	Successful	Unsuccessful on syntax error
5.Drop	Drop table	Successful	Unsuccessful on syntax error

FAQs:

1. Explain the types of DML commands?

2.What is operators, function, set-operators ?

3. What is difference between delete and truncate commands?

4. What is difference between primary key and unique key?

5. Explain different set operations commands in Mysql?

Oral/Review Questions:

1. List types of comparison operators?

2. What are DCL and TCL commands?

ASSIGNMENT NUMBER: A4

Revised On: 15/06/2018

TITLE	Design at least 10 SQL queries for suitable database application using SQL DML statements:All types of join, sub-query and View.
PROBLEM STATEMENT /DEFINITION	Design at least 10 SQL queries for suitable database application using SQL DML statements:All types of join, sub-query and View
OBJECTIVE	To understand • Types of joins. • Subquery and its types. • Complex views
S/W PACKAGES AND HARDWARE APPARATUS USED	MY-SQL PC with the configuration as Latest Version of 64 bit Operating Systems, Open Source Fedora-GHz. 8 G.B. RAM, 500 G.B. HDD, 15"Color Monitor, Keyboard, Mouse
REFERENCES	• Maurice J.Bach "The Design of The Unix Operating system", PHI, ISBN 978-81-203-0516-8 • Evi Nemeth, Garth Snyder, Tren Hein, Ben Whaley, Unix and Linux System • Administration Handbook, Fourth Edition, ISBN: 978-81-317-6177-9, 2011

STEPS	Refer to details
INSTRUCTIONS FOR WRITING JOURNAL	• Title • Problem Definition • Learning Objectives • Theory • Class Diagram/ER Diagram • Test cases • Program Listing • Output • Conclusion

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

Objective:to understand all types of Joins in SQL, Sub queries, operations on views such as insert, update, delete

Relevant Theory

❖ **Joins:**

"JOIN" is an SQL keyword used to query data from two or more related tables.
The join operation allows the combining of two relations by merging pairs of tuples, one from each relation, into a single tuple.

In a database such as MySQL, data is divided into a series of tableswhich are then connected together in SELECT commands to generate the output required.

There are a number of different ways to join relations such as

- The Natural Join or Regular Join
- Inner Join
- Left outer join
- Right Outer join
- Full outer join

We will see how these join works in MySQL

The Natural Join or Regular Join

The natural join operation operates on two relations and produces a relationas the result.

Natural join considersonly those pairs of tuples with the same value on those attributes that appear inthe schemas of both relations.

For example consider following two relations

ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

Instructor Relegation

ID	*course_id*	*sec_id*	*semester*	*year*
10101	CS-101	1	Fall	2009
10101	CS-315	1	Spring	2010
10101	CS-347	1	Fall	2009
12121	FIN-201	1	Spring	2010
15151	MU-199	1	Spring	2010
22222	PHY-101	1	Fall	2009
32343	HIS-351	1	Spring	2010
45565	CS-101	1	Spring	2010
45565	CS-319	1	Spring	2010
76766	BIO-101	1	Summer	2009
76766	BIO-301	1	Summer	2010
83821	CS-190	1	Spring	2009
83821	CS-190	2	Spring	2009
83821	CS-319	2	Spring	2010
98345	EE-181	1	Spring	2009

Teaches Relation

Computing **instructor natural join teaches** considers onlythose pairs of tuples where both the tuple from instructor and the tuple fromteaches have the same value on the common attribute, ID.

The result relation, shown in below, has only 13 tuples, the ones thatgive information about an instructor and a course that that instructor actuallyteaches. Notice that we do not repeat those attributes that appear in the schemasof both relations; rather they appear only once.

Notice also the order in which theattributes are listed: first the attributes common to the schemas of both relations,second those attributes unique to the schema of the first relation, and finally, thoseattributes unique to the schema of the second relation.

ID	name	dept_name	salary	course_id	sec_id	semester	year
10101	Srinivasan	Comp. Sci.	65000	CS-101	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	CS-315	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	CS-347	1	Fall	2009
12121	Wu	Finance	90000	FIN-201	1	Spring	2010
15151	Mozart	Music	40000	MU-199	1	Spring	2010
22222	Einstein	Physics	95000	PHY-101	1	Fall	2009
32343	El Said	History	60000	HIS-351	1	Spring	2010
45565	Katz	Comp. Sci.	75000	CS-101	1	Spring	2010
45565	Katz	Comp. Sci.	75000	CS-319	1	Spring	2010
76766	Crick	Biology	72000	BIO-101	1	Summer	2009
76766	Crick	Biology	72000	BIO-301	1	Summer	2010
83821	Brandt	Comp. Sci.	92000	CS-190	1	Spring	2009
83821	Brandt	Comp. Sci.	92000	CS-190	2	Spring	2009
83821	Brandt	Comp. Sci.	92000	CS-319	2	Spring	2010
98345	Kim	Elec. Eng.	80000	EE-181	1	Spring	2009

The natural join of the instructor relation with the teaches relation.

Consider the query “For all instructors in the university who have taught some course, find their names and the course ID of all courses they taught”, which we wrote as:
select name, course id
from instructor, teaches
where instructor.ID= teaches.ID;
This query can be written more concisely using the natural-join operation inSQL as:
select name, course id
from instructor natural join teaches;

To understand how to compute **natural join in MySQL**consider below sample data such as :
Mr. Brown, Person number 1, has a phone number 01225 708225
Miss Smith, Person number 2, has a phone number 01225 899360
Mr. Pullen, Person number 3, has a phone number 01380 724040
and also:
Person number 1 is selling property number 1 - Old House Farm
Person number 3 is selling property number 2 - The Willows
Person number 3 is (also) selling property number 3 - Tall Trees
Person number 3 is (also) selling property number 4 - The Melksham Florist

Person number 4 is selling property number 5 - Dun Roamin.

Create two tables such as **demo_people** and **demo_property** and insert above data in to it. Then after select all data from both the tables using below expressions

mysql> select * from demo_people;

```
+------------+--------------+------+
| name       | phone        | pid  |
+------------+--------------+------+
| Mr Brown   | 01225 708225 |    1 |
| Miss Smith | 01225 899360 |    2 |
| MrPullen   | 01380 724040 |    3 |
+------------+--------------+------+
```

3 rows in set (0.00 sec)

mysql> select * from demo_property;

```
+------+------+----------------------+
| pid  |spid | selling              |
+------+------+----------------------+
|    1 |    1 | Old House Farm       |
|    3 |    2 | The Willows          |
|    3 |    3 | Tall Trees           |
```

| 3 | 4 | The Melksham Florist |

| 4 | 5 | Dun Roamin |

+------+------+----------------------+

5 rows in set (0.00 sec)

mysql>

Natural join or Regular join on above both tables will be done by following query expression

mysql> select name, phone, selling

from demo_people**join**demo_property

on demo_people.pid = demo_property.pid;

+-----------+--------------+----------------------+

| name | phone | selling |

+-----------+--------------+----------------------+

| MrBrown | 01225 708225 | Old House Farm |

| Mr Pullen | 01380 724040 | The Willows |

| Mr Pullen | 01380 724040 | Tall Trees |

| Mr Pullen | 01380 724040 | The Melksham Florist |

+-----------+--------------+----------------------+

4 rows in set (0.01 sec)

mysql>

It will give all records that match the value of appropriate attributesin the two tables, and records in both incoming tables that do not match are discarded.

Such types of join operation that do not preserve non-matched tuples are called as **inner join operations**.

Outer Join

An outer join does not require each record in the two joined tables to have a matching record or attribute. The joined table retains each record—even if no other matching record exists.
There are in fact three forms of outer join:

- The **left outer join** preserves tuples only in the relation named before (to theleft of) the left outer join operation.
- The **right outer join** preserves tuples only in the relation named after (to theright of) the right outer join operation.
- The **full outer join** preserves tuples in both relations. (In this case left and right refer to the two sides of the JOIN keyword.)

No implicit join-notation for outer joins exists in standard SQL.

Left Outer Join

The result of a left outer join for tables A and B always contains all records of the "left" table (table A), even if the join-condition does not find any matching record in the "right" table (tableB).
This means that if the ON clause matches zero records in B for a given record in A, the join will still return a row in the result for that record—but with NULL in each column from B.
A left outer join returns all the values from an inner join plus all values in the left table that do not match to the right table, including rows with NULL (empty) values in the link field.
To understand how to compute **left outer join in MySQL** again consider **demo_people** and **demo_property**tables
After computing LEFT JOIN, result contains all records that match in the same way and IN ADDITION it will shows an extra record for each unmatched record in the left table of the join - thus ensuring that every PERSON is included in result generated by left outer join:

```
mysql> select name, phone, selling

fromdemo_people left joindemo_property

on demo_people.pid = demo_property.pid;

+------------+--------------+----------------------+

| name       | phone        | selling              |
```

```
+------------+--------------+---------------------+
| Mr Brown   | 01225 708225 | Old House Farm      |
| Miss Smith | 01225 899360 | NULL                |
| MrPullen   | 01380 724040 | The Willows         |
| MrPullen   | 01380 724040 | Tall Trees          |
| MrPullen   | 01380 724040 | The Melksham Florist |
+------------+--------------+---------------------+
5 rows in set (0.00 sec)

mysql>
```

Right Outer Join

The result of a right outer join for tables A and B always contains all records of the "right" table (table B), even if the join-condition does not find any matching record in the "Left" table (table A).

This means that if the ON clause matches zero records in A for a given record in B, the join will still return a row in the result for that record—but with NULL in each column from A.

A right outer join returns all the values from an inner join plus all values in the right table that do not match to the left table, including rows with NULL (empty) values in the link field.

To understand how to compute **right outer join in MySQL** again consider **demo_people** and **demo_property**tables

After computing RIGHT JOIN, result contains all the records that match and IN ADDITION it will show extra record for each unmatched record in the right table of the join - that means that each property gets a mention even if we don't have seller details:

```
mysql> select name, phone, selling
```

fromdemo_people **right join** demo_property

on demo_people.pid = demo_property.pid;

```
+-----------+--------------+----------------------+
| name      | phone        | selling              |
+-----------+--------------+----------------------+
| MrBrown   | 01225 708225 | Old House Farm       |
| Mr Pullen | 01380 724040 | The Willows          |
| Mr Pullen | 01380 724040 | Tall Trees           |
| Mr Pullen | 01380 724040 | The Melksham Florist |
| NULL      | NULL         | Dun Roamin           |
+-----------+--------------+----------------------+
```

5 rows in set (0.00 sec)

mysql>

Full Outer Join

The full outer join is a combination of the left and right outer-join types. After the operation computes the result of the inner join, it extends with nulls those tuples from the left-hand-side relation that did not match with any from the right-hand side relation, and adds them to the result.
Similarly, it extends with nulls those tuples from the right-hand-side relation that did not match with any tuples from the left-hand-side relation and adds them to the result.

A standard SQL FULL OUTER join is like a LEFT or RIGHT join, except that it includes all rows from both tables, matching them where possible and filling in with NULLs where there is no match.Here are two tables, apples and oranges:

Apples

Variety	**Price**
Fuji	5.00
Gala	6.00

Join them on price. Here is the left join:

select * from apples as a

Oranges	
Variety	**Price**
Valencia	4.00
Navel	5.00

left outer join oranges as o on a.price = o.price

Variety	**price**	**variety**	**price**
Fuji	5	Navel	5
Gala	6	NULL	NULL

And the right joins:

select * from apples as a
right outer join oranges as o on a.price = o.price

Variety	**price**	**variety**	**price**

NULL	NULL	Valencia	4
Fuji	5	Navel	5

The FULL OUTER JOIN of these two tables, on price, should give the following result:

Variety	**price**	**variety**	**price**
Fuji	5	Navel	5
Gala	6	NULL	NULL
NULL	NULL	Valencia	4

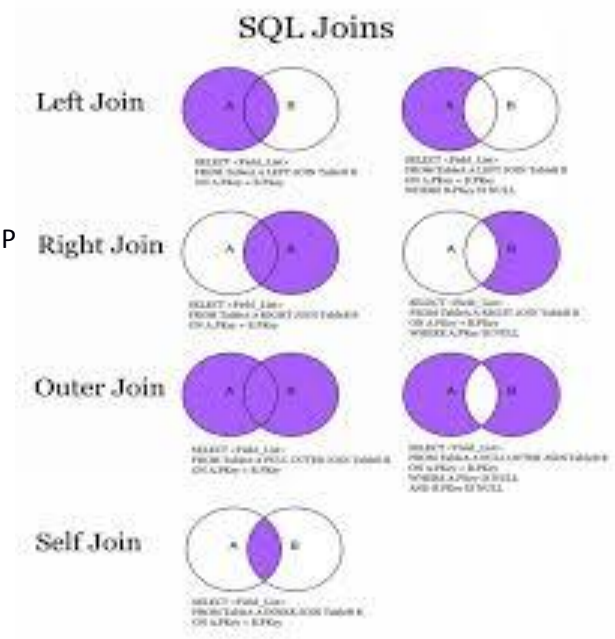
Here is a script to create and populate the example tables, so you can follow along:

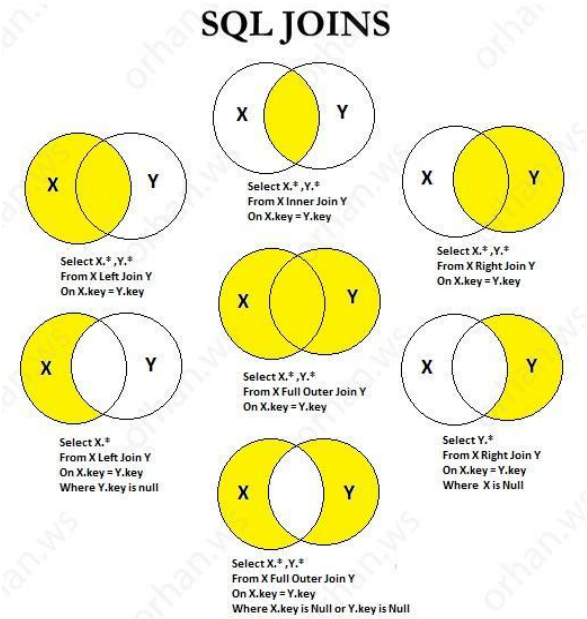
Mysql>create table apples (variety char(10) not null primary key, price int not null);
Mysql>create table oranges (variety char(10) not null primary key, price int not null);
Mysql>insert into apples(variety, price) values('Fuji',5),('Gala',6);
Mysql>insert into oranges(variety, price) values('Valencia',4),('Navel',5);

One method to simulate a full outer join is to take the union of two outer joins, for example,

Mysql>select * from apples as a
left outer join oranges as o on a.price = o.price
union
select * from apples as a
right outer join oranges as o on a.price = o.price

This gives the desired results in this case.



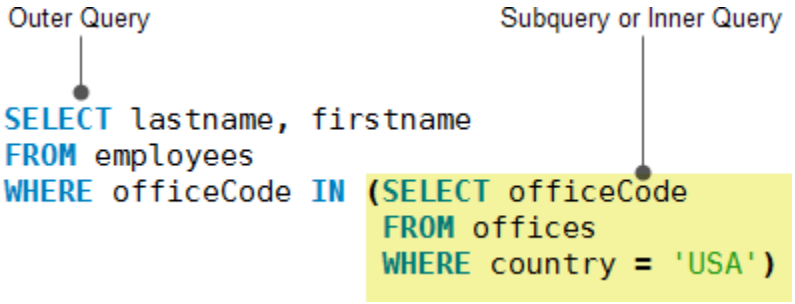


❖ **Sub Queries**

A MySQL subquery is a query that is nested inside another query such as SELECT, INSERT, UPDATE or DELETE. A MySQL subquery is also can be nested inside another subquery. A MySQL subquery is also called an inner query, while the query that contains the subquery is called an outer query.

For example following subquery that returns employees who locate in the offices in the USA.

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



You can use a subquery anywhere an expression can be used. A subquery also must be enclosed in parentheses.

MySQL subquery within a WHERE clause

1. MySQL subquery with comparison operators

If a subquery returns a single value, you can use comparison operators to compare it with the expression in the WHERE clause.

For example, the following query returns the customer who has the maximum payment.

```
Mysql> SELECT customerNumber, checkNumber, amount
FROM payments
WHERE amount = (
   SELECT MAX(amount)
      FROM payments
)
```

Other comparison operators can be used such as greater than (>), less than(<), etc.
For example, you can find customer whose payment is greater than the average payment.
A subquery is used to calculate the average payment by using the AVG aggregate function.
The outer query selects payments that are greater than the average payment returned from the subquery.

```
Mysql> SELECT customerNumber,
      checkNumber,
      amount
FROM payments
WHERE amount > (
   SELECT AVG(amount)
   FROM payments
)
```

2. MySQL subquery with IN and NOT IN operators

If a subquery returns more than one value, you can use other operators such as IN or NOT IN operator in the WHERE clause.
For example, you can use a subquery with NOT IN operator to find customer who has not ordered any product as follows:

```
Mysql> SELECT customername
FROM customers
WHERE customerNumber NOT IN (
   SELECT DISTINCT customernumber
   FROM orders
)
```

3. MySQL subquery with EXISTS and NOT EXISTS

When a subquery is used with EXISTS or NOT EXISTS operator, a subquery returns a Boolean value of TRUE or FALSE. The subquery acts as an existence check.

In the following example, we select a list of customers who have at least one order with total sales greater than 10K.
First, we build a query that checks if there is at least one order with total sales greater than 10K:

```
Mysql > SELECT priceEach * quantityOrdered
FROM orderdetails
WHERE priceEach * quantityOrdered> 10000
GROUP BY orderNumber
```

The query returns some records so that when we use it as a subquery, it will return TRUE; therefore the whole query will return all customers:

```
Mysql > SELECT customerName
FROM customers
WHERE EXISTS (
   SELECT priceEach * quantityOrdered
   FROM orderdetails
   WHERE priceEach * quantityOrdered> 10000
   GROUP BY orderNumber
)
```

If you replace the EXISTS by NOT EXIST in the query, it will not return any record at all.

4. MySQL subquery in FROM clause

When you use a subquery in the FROM clause, the result set returned from a subquery is used as a table. This table is referred to as a derived table or materialized subquery.
The following subquery finds the maximum, minimum and average number of items in sale orders:

```
SELECT max(items),
      min(items),
      floor(avg(items))
FROM
(SELECT orderNumber,
   count(orderNumber) AS items
FROM orderdetails
GROUP BY orderNumber) AS lineitems
```

❖ **View:**

MySQL supports database views or views since version 5.X. In MySQL, almost features of views conform to the SQL: 2003 standard. MySQL process queries to the views in two ways:

- MySQL creates a temporary table based on the view definition statement and then executes the incoming query on this temporary table.
- First, MySQL combines the incoming query with the query defined the view into one query. Then, MySQL executes the combined query.

MySQL supports version system for views. Each time when the view is altered or replaced, a copy of the existing view is back up in arc (archive) folder which resides in a specific database folder.
The name of back up file is view_name.frm-00001. If you then change the view again, MySQL will create a new backup file named view_name.frm-00002.
MySQL also allows you to create a view of views. In the SELECT statement of view definition, you can refer to another view.
If the cache is enabled, the query against a view is stored in the cache. As the result, it increases the performance of the query by pulling data from the cache instead of querying data from the underlying tables.

Creating Updateable Views

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

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

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

Example of creating updateable view

Create a view named officeInfo against the offices table. The view refers to three columns of the offices table: officeCode, phone and city.

```
Mysql> CREATE VIEW officeInfo
 AS
   SELECT officeCode, phone, city
   FROM offices
```

We can query data from the officeInfo view using the SELECT statement.

```
Mysql> SELECT * FROM officeInfo
```

Then, we can change the phone number of the office with officeCode 4 through theofficeInfo view by using the UPDATE statement.

```
Mysql> UPDATE officeInfo
SET phone = '+33 14 723 5555'
WHERE officeCode = 4
```

Finally, to see the change, we can select the data from the officeInfo view by executing following query:

```
My sql> SELECT * FROM officeInfo
WHERE officeCode = 4
```

Managing Views in MySQL: Modifying views

Once a view is defined, can be modified by using the ALTER VIEW statement.
The syntax of the ALTER VIEW statement is similar to the CREATE VIEW statement except the CREATE keyword is replaced by the ALTER keyword.

```
ALTER
 [ALGORITHM =  {MERGE | TEMPTABLE | UNDEFINED}]
  VIEW [database_name].  [view_name]
   AS
 [SELECT statement]
```

The following query modifies the organization view by adding an addition email field.

```
Mysql> ALTER VIEW organization
  AS
  SELECT CONCAT(E.lastname,E.firstname) AS Employee,
        E.email AS  employeeEmail,
        CONCAT(M.lastname,M.firstname) AS Manager
  FROM employees AS E
  INNER JOIN employees AS M
    ON M.employeeNumber = E.ReportsTo
  ORDER BY Manager
```

To verify the change, you can query data from the organization view:

ELECT * FROM Organization

MySQL drop views

Once a view created, you can remove it by using the DROP VIEW statement. The following illustrates the syntax of the DROP VIEW statement:

DROP VIEW [IF EXISTS] [database_name].[view_name]

The IF EXISTS is the optional element of the statement, which allows you to check whether the view exists or not. It helps you avoid an error of removing a non-existent view.

For example, if you want to remove the organization view, you can use the DROP VIEW statement as follows:

DROP VIEW IF EXISTS organization

Each time you modify or remove a view, MySQL makes a back up of the view definition file to the/database_name/arc/ folder. In case you modify or remove a view by accident, you can get a back up from there.

Exercises:

Refer Sample Database: University Database provided with assignment

1. For all instructors in the university who have taught some course, find their names and the course ID of all courses they taught using natural join.
2. List the names of instructors along with the titles of courses that they teach using natural join.
3. Find all the courses taught **in the both** the Fall 2009 and Spring 2010 semesters using subqueries.
4. Find all the courses taught in the Fall 2009 semester **but not in** the Spring 2010 semester using subqueries.
5. Find the total number of (distinct) students who have taken course sections taught by the instructor with ID 110011.
6. Find the names of all instructors whose salary is greater than at least one instructor in the Biology department.
7. Find the names of all instructors that have a salary value greater than that of each instructor in the Biology department.
8. Find the departments that have the highest average salary.
9. Find all students who have taken all courses offered in the Biology department.
10. Compute left outer join between students and takes relation.
11. Find all students who have not taken a course using left outer join.
12. Display a list of all students in the Comp. Sci. department, along with the course sections, if any, that they have taken in Spring 2009; all course sections from Spring 2009 must be displayed, even if no student from the Comp. Sci. department has taken the course section.

13. Create a view as faculty to access all the data from instructor relation except salary.
14. Create a view that lists all course sections offered by the Physics department in the Fall 2009 semester with the building and room number of each section.

Conclusion:

- We have explained the MySQL LEFT JOIN, RIGHT JOIN clause and shown you how to apply it to query data from multiple database tables.
- We have shown you how to use MySQL subquery to write more complex queries.
- We have shown you how to create an updateable view and how to update data in the underlying table through the view.
- We have learned how to manage views in MySQL including displaying, modifying and removing views.

FAQs:

1. What is impotance of JOINS in Mysql?
2. Explain foreign key constraints worth suitable example?
3. What is difference between left outer anf right outer joins?
4. What is join? Explain different types of join
5. What is sub-query?
6. Is view is virtual justify your answer?

Oral/Review Questions:

1. Give difference between natural and inner join?
2. Give difference between left-outer and right-outer join?
3. What is nested query?
4. What is self join?
5. Explain equi JOIN and non-equi JOIN?

ASSIGNMENT NUMBER: A5

Revised On: 15/06/2018

TITLE	Write a PL/SQL block of code for the given requirements
PROBLEM STATEMENT /DEFINITION	Write a PL/SQL block of code for the following requirements
OBJECTIVE	• To understand the control structure • To understand exception handling in PL/SQL
S/W PACKAGES AND HARDWARE APPARATUS USED	SQL package PC with the configuration as Latest Version of 64 bit Operating Systems, Open Source Fedora-GHz. 8 G.B. RAM, 500 G.B. HDD, 15"Color Monitor, Keyboard, Mouse
REFERENCES	Maurice J.Bach "The Design of The Unix Operating system", PHI, ISBN 978-81-203-0516-8 Evi Nemeth, Garth Snyder, Tren Hein, Ben Whaley, Unix and Linux System Administration Handbook, Fourth Edition, ISBN: 978-81-317-6177-9, 2011
STEPS	Refer to details
INSTRUCTIONS FOR WRITING	• **Title** • **Problem Definition**

JOURNAL	• **Learning Objectives** • **Theory** • **Class Diagram/ER Diagram** • **Test cases** • **Program Listing** • **Output** • **Conclusion**

Aim: Write a PL/SQL block of code for the following requirements:-

Schema:

Customer(Cust_id,Name, DateofPayment, NameofScheme, Status)

Fine(Cust_id, Date, Amt)

1. Accept Cust_id & name of scheme from user.
2. Check the number of days (from date of payment), if days are between 15 to 30 then fine amount will be Rs 5per day.
3. If no. of days>30, per day fine will be Rs 50 per day & for days less than 30, Rs. 5 per day.
4. After payment, status will change from N to P.

If condition of fine is true, then details will be stored into Fine table.

Pre-requisite:

Basic knowledge of exception handling and control structure

Learning Objectives:

- To understand and write PL/SQL block code requirements defined.
- To understand exception handling.
- To understand basic structure of PL/SQL block
- To apply control structure

Learning Outcomes:

The students will be able to implement

- PL/SQL block, user-defined and predefined exception handling.
- Control structure using PL/SQL.

Theory:

PL/SQL:

PL/SQL stands for *Procedural Language/Structured Query Language*. PL/SQL offers a set of procedural commands (IF statements, loops, assignments), organized within blocks (explained below), that complement and extend the reach of SQL.

BLOCKS in PL\SQL:

PL/SQL is a block-structured language. A PL/SQL block is defined by the keywords DECLARE, BEGIN, EXCEPTION, and END, which break up the block into three sections:

1. **Declarative**: statements that declare variables, constants, and other code elements, which can then be used within that block
2. **Executable**: statements that are run when the block is executed
3. **Exception handling**: a specially structured section you can use to "catch," or trap, any exceptions that are raised when the executable section runs.

Some examples:

- The classic "Hello World!" block contains an executable section that calls the DBMS_OUTPUT.PUT_LINE procedure to display text on the screen:

```
BEGIN
  DBMS_OUTPUT.put_line ('Hello World!');
END;
```

EXCEPTION HANDLING:

PL/SQL provides a feature to handle the Exceptions which occur in a PL/SQL Block known as exception Handling. Using Exception Handling we can test the code and avoid it from exiting abruptly. When an exception occurs a messages which explains its cause is received. PL/SQL Exception message consists of three parts.

1) Type of Exception
2) An Error Code
3) A message

Structure of Exception Handling.

General Syntax for coding the exception section

```
DECLARE

  Declaration section
```

```
BEGIN
  Exception section
EXCEPTION
WHEN ex_name1 THEN
   -Error handling statements
WHEN ex_name2 THEN
   -Error handling statements
WHEN Others THEN
  -Error handling statements
END;
```

General PL/SQL statements can be used in the Exception Block.

When an exception is raised, Oracle searches for an appropriate exception handler in the exception section. For example in the above example, if the error raised is 'ex_name1 ', then the error is handled according to the statements under it. Since, it is not possible to determine all the possible run time errors during testing fo the code, the 'WHEN Others' exception is used to manage the exceptions that are not explicitly handled. Only one exception can be raised in a Block and the control does not return to the Execution Section after the error is handled.

If there are nested PL/SQL blocks like this.

```
DELCARE
  Declaration section
BEGIN
   DECLARE
```

```
    Declaration section
  BEGIN
    Execution section
  EXCEPTION
    Exception section
  END;
EXCEPTION
  Exception section
END;
```

In the above case, if the exception is raised in the inner block it should be handled in the exception block of the inner PL/SQL block else the control moves to the Exception block of the next upper PL/SQL Block. If none of the blocks handle the exception the program ends abruptly with an error.

Types of Exception.

There are 3 types of Exceptions.
a) Named System Exceptions
b) Unnamed System Exceptions
c) User-defined Exceptions

a) Named System Exceptions

System exceptions are automatically raised by Oracle, when a program violates a RDBMS rule. There are some system exceptions which are raised frequently, so they are pre-defined and given a name in Oracle which are known as Named System Exceptions.

For example: NO_DATA_FOUND and ZERO_DIVIDE are called Named System exceptions.

Named system exceptions are:
1) Not Declared explicitly,
2) Raised implicitly when a predefined Oracle error occurs,
3) caught by referencing the standard name within an exception-handling routine.

b) Unnamed System Exceptions

Those system exception for which oracle does not provide a name is known as unamed system exception. These exception do not occur frequently. These Exceptions have a code and an associated message.

There are two ways to handle unnamed sysyem exceptions:
1. By using the WHEN OTHERS exception handler, or
2. By associating the exception code to a name and using it as a named exception.

We can assign a name to unnamed system exceptions using a Pragma called EXCEPTION_INIT. EXCEPTION_INIT will associate a predefined Oracle error number to a programmer_defined exception name.

Steps to be followed to use unnamed system exceptions are
• They are raised implicitly.
• If they are not handled in WHEN Others they must be handled explicity.
• To handle the exception explicity, they must be declared using Pragma EXCEPTION_INIT as given above and handled referecing the user-defined exception name in the exception section.

The general syntax to declare unnamed system exception using EXCEPTION_INIT is:

```
DECLARE

   exception_name EXCEPTION;

   PRAGMA

   EXCEPTION_INIT (exception_name, Err_code);

BEGIN
```

Execution section

EXCEPTION

WHEN exception_name THEN

handle the exception

END;

For Example: Lets consider the product table and order_items table from sql joins.

Here product_id is a primary key in product table and a foreign key in order_items table.
If we try to delete a product_id from the product table when it has child records in order_id table an exception will be thrown with oracle code number -2292.
We can provide a name to this exception and handle it in the exception section as given below.

```
DECLARE
 Child_rec_exception EXCEPTION;
 PRAGMA
  EXCEPTION_INIT (Child_rec_exception, -2292);
BEGIN
 Delete FROM product where product_id= 104;
EXCEPTION
  WHEN Child_rec_exception
  THEN Dbms_output.put_line('Child records are present for this product_id.');
END;
```

c) User-defined Exceptions

Apart from sytem exceptions we can explicity define exceptions based on business rules. These are known as user-defined exceptions.

Steps to be followed to use user-defined exceptions:
• They should be explicitly declared in the declaration section.
• They should be explicitly raised in the Execution Section.
• They should be handled by referencing the user-defined exception name in the exception section.

For Example: Lets consider the product table and order_items table from sql joins to explain user-defined exception. Lets create a business rule that if the total no of units of any particular product sold is more than 20, then it is a huge quantity and a special discount should be provided.

```
DECLARE
  huge_quantity EXCEPTION;
  CURSOR product_quantity is
  SELECT p.product_name as name, sum(o.total_units) as units
  FROM order_tems o, product p
  WHERE o.product_id = p.product_id;
  quantity order_tems.total_units%type;
  up_limit CONSTANT order_tems.total_units%type := 20;
  message VARCHAR2(50);
BEGIN
  FOR product_rec in product_quantity LOOP
```

```
   quantity := product_rec.units;
    IF quantity > up_limit THEN
     message := 'The number of units of product ' || product_rec.name ||
                ' is more than 20. Special discounts should be provided.
                Rest of the records are skipped. '
    RAISE huge_quantity;
    ELSIF quantity < up_limit THEN
     v_message:= 'The number of unit is below the discount limit.';
    END IF;
    dbms_output.put_line (message);
 END LOOP;
EXCEPTION
  WHEN huge_quantity THEN
    dbms_output.put_line (message);
END;
```

FAQs:

1. What is basic difference between sql and PLSQL?
2. Explain different control structures in PLSQL with suitable example?
3. What is PL-SQL block? Give types of PL-SQL block?
4. What is user defined and pre-defined exceptions?
5. What is cursors? Explain different types of cursors with suitable example?

Oral/Review Questions:

1. What are the different control structures supported in PL-SQL ?
2. What are the data types in PL-SQL?
3. Explain basic structure of PL-SQL?

ASSIGNMENT NUMBER: A6

Revised On: 15/06/2018

TITLE	Write a PL/SQL block of code using parameterized Cursor, that will merge the data available in the newly created table
PROBLEM STATEMENT /DEFINITION	Write a PL/SQL block of code using parameterized Cursor, that will merge the data available in the newly created table
OBJECTIVE	• To understand the types of cursors • To understand how to use cursors with PL/SQL block
S/W PACKAGES AND HARDWARE APPARATUS USED	SQL package PC with the configuration as Latest Version of 64 bit Operating Systems,Open Source Fedora-GHz. 8 G.B. RAM, 500 G.B. HDD, 15"Color Monitor, Keyboard, Mouse
REFERENCES	Maurice J.Bach "The Design of The Unix Operating system", PHI, ISBN 978-81-203-0516-8 Evi Nemeth, Garth Snyder, Tren Hein, Ben Whaley, Unix and Linux System Administration Handbook, Fourth Edition, ISBN: 978-81-317-6177-9, 2011
STEPS	Refer to details
INSTRUCTIONS FOR WRITING	• Date • Title

JOURNAL	• Problem Definition • Learning Objective • Learning Outcome • Theory-Related concept,Architecture,Syntax etc • Class Diagram/ER diagram • Test cases • Program Listing • Output • Conclusion

Aim: 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_EmpId with the data available in the table O_EmpId.

If the data in the first table already exist in the second table then that data should be skipped.

Pre-requisite:

Basic knowledge of MY-SQL and cursors

Learning Objectives:

- To understand and implement types of cursors with PL/SQL block code.

Learning Outcomes:

The students will be able to

- Implement PL/SQL block code.
- Implement types of curssors

Theory:

PL/SQL:

PL/SQL stands for *Procedural Language/Structured Query Language*. PL/SQL offers a set of procedural commands (IF statements, loops, assignments), organized within blocks (explained below), that complement and extend the reach of SQL.

BLOCKS in PL\SQL:

PL/SQL is a block-structured language. A PL/SQL block is defined by the keywords DECLARE, BEGIN, EXCEPTION, and END, which break up the block into three sections:

4. **Declarative**: statements that declare variables, constants, and other code elements, which can then be used within that block
5. **Executable**: statements that are run when the block is executed
6. **Exception handling**: a specially structured section you can use to "catch," or trap, any exceptions that are raised when the executable section runs.

Some examples:

- The classic "Hello World!" block contains an executable section that calls the DBMS_OUTPUT.PUT_LINE procedure to display text on the screen:

```
BEGIN
  DBMS_OUTPUT.put_line ('Hello World!');
END;
```

CURSORS:

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. A cursor can hold more than one row, but can process only one row at a time. The set of rows the cursor holds is called the active set.

There are two types of cursors in PL/SQL:

Implicit cursors

These are created by default when DML statements like, INSERT, UPDATE, and DELETE statements are executed. They are also created when a SELECT statement that returns just one row is executed.

Explicit cursors

They must be created when you are executing a SELECT statement that returns more than one row. Even though the cursor stores multiple records, only one record can be processed at a time, which is called as current row. When you fetch a row the current row position moves to next row.

Both implicit and explicit cursors have the same functionality, but they differ in the way they are accessedfor Example: Consider the PL/SQL Block that uses implicit cursor attributes as shown below:

```
DECLARE  var_rows number(5);

BEGIN

  UPDATE employee

  SET salary = salary + 1000;

  IF SQL%NOTFOUND THEN
```

```
  dbms_output.put_line('None of the salaries where updated');
ELSIF SQL%FOUND THEN
  var_rows := SQL%ROWCOUNT;
  dbms_output.put_line('Salaries for ' || var_rows || 'employees are updated');
END IF;
END;
```

In the above PL/SQL Block, the salaries of all the employees in the 'employee' table are updated. If none of the employee's salary are updated we get a message 'None of the salaries where updated'. Else we get a message like for example, 'Salaries for 1000 employees are updated' if there are 1000 rows in 'employee' table.

FAQs:

1. What is importance of cursor in PLSQL?
2. Explain different types of implicit cursor with suitable examples?
3. What are cursors?
4. Explain different types of cursors with suitable examples?
5. What is mean by parameterized cursors?

Oral/Review Questions:

1. Give difference between implicit and explicit cursors?
2. Explain steps for using cursor?
3. Explain significance of using cursors?

ASSIGNMENT NUMBER: A7

Revised On: 15/06/2018

TITLE	PL/SQL Stored Procedure and Stored Function
PROBLEM STATEMENT /DEFINITION	Write a Stored Procedure namely proc_Grade for the categorization of student.
OBJECTIVE	• Understand the PL/SQL Stored Procedure. • Understand the PL/SQL Stored Function • Write PL/SQL block code using stored procedure and stored function.
S/W PACKAGES AND HARDWARE APPARATUS USED	• Mysql • PL/SQL
REFERENCES	1 Silberschatz A., Korth H., Sudarshan S., "Database System Concepts", 5th Edition, McGraw Hill Publishers, 2002, ISBN 0-07-120413-X 1. Connally T., Begg C., "Database Systems", 3rd Edition, Pearson Education, 2002, ISBN 81-7808-861-4 2. mysql.com/docs/mysql-tutorial-excerpt-5.1-en.pdf
STEPS	**Refer to details**
INSTRUCTIONS FOR WRITING JOURNAL	• Date • Title • Problem Definition • Learning Objectives • Learning Outcomes • Theory

	• Class Diagram/ER Diagram • Test cases • Program Listing • Output • Conclusion

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

Write a Stored Procedure namely proc_Grade for the categorization of customer. If purchase by customer in year is <=20000 and >=10000 then customer will be placed in platinum category. If purchase by customer is between 9999 and 5000 category is gold, if purchase between 4999 and 2000 category is silver.

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

Customer(Cust_id,name, total_purchase)

Category(Cust_id,Name,Class)

Aim: PL/SQL Stored Procedure and Stored Function

Pre-requisite:

Basic knowledge of PL/SQL Commands and Mysql.

Learning Objectives:

- To understand & implement the stored function and stored procedures in PL/SQL.
- To pass in, out parameters for function and procedure.

Learning Outcomes:

The students will be able to

- Implement stored function.
- Implement stored procedure

Theory:

PL/SQL:

PL/SQL stands for *Procedural Language/Structured Query Language*. PL/SQL offers a set of procedural commands (IF statements, loops, assignments), organized within blocks (explained below), that complement and extend the reach of SQL.

BLOCKS in PL\SQL:

PL/SQL is a block-structured language. A PL/SQL block is defined by the keywords DECLARE, BEGIN, EXCEPTION, and END, which break up the block into three sections:

1. **Declarative**: statements that declare variables, constants, and other code elements, which can then be used within that block
2. **Executable**: statements that are run when the block is executed
3. **Exception handling**: a specially structured section you can use to "catch," or trap, any exceptions that are raised when the executable section runs.

Some examples:

- The classic "Hello World!" block contains an executable section that calls the DBMS_OUTPUT.PUT_LINE procedure to display text on the screen:

```
BEGIN
  DBMS_OUTPUT.put_line ('Hello World!');
END;
```

A) Stored Procedures:

1) What is a Stored Procedure?

A **stored procedure** or in simple a **proc** is a named PL/SQL block which performs one or more specific task. This is similar to a procedure in other programming languages. A

procedure has a header and a body. The header consists of the name of the procedure and the parameters or variables passed to the procedure. The body consists or declaration section, execution section and exception section similar to a general PL/SQL Block. A procedure is similar to an anonymous PL/SQL Block but it is named for repeated usage.

Procedures: Passing Parameters

We can pass parameters to procedures in three ways.

1) IN-parameters
2) OUT-parameters
3) IN OUT-parameters

A procedure may or may not return any value.

General Syntax to create a procedure is:

CREATE [OR REPLACE] PROCEDURE proc_name [list of parameters]

IS

Declaration section

BEGIN

Execution section

EXCEPTION

Exception section

END;

IS - marks the beginning of the body of the procedure and is similar to DECLARE in anonymous PL/SQL Blocks. The code between IS and BEGIN forms the Declaration section.

The syntax within the brackets [ ] indicate they are optional. By using CREATE OR REPLACE together the procedure is created if no other procedure with the same name exists or the existing procedure is replaced with the current code.

How to execute a Stored Procedure?

There are two ways to execute a procedure.

1) From the SQL prompt.

EXECUTE [or EXEC] procedure_name;

2) Within another procedure – simply use the procedure name.

procedure_name;

NOTE: In the examples given above, we are using backward slash '/' at the end of the program. This indicates the oracle engine that the PL/SQL program has ended and it can begin processing the statements.

B)Stored Function:

What is a Function in PL/SQL?

A function is a named PL/SQL Block which is similar to a procedure. The major difference between a procedure and a function is, a function must always return a value, but a procedure may or may not return a value.

General Syntax to create a function is

CREATE [OR REPLACE] FUNCTION function_name [parameters]

RETURN return_datatype;

IS

Declaration_section

BEGIN

Execution_section

Return return_variable;

EXCEPTION

exception section

Return return_variable;

END;

1) **Return Type:** The header section defines the return type of the function. The return datatype can be any of the oracle datatype like varchar, number etc.
2) The execution and exception section both should return a value which is of the datatype defined in the header section.

FAQ:

1. What is significance of stored procedure ?
2. What is stored procedure?
3. Give difference between stored functions and stored procedure?

Oral/Review questions:

1. What are the advantages of using stored procedure?
2. Explain the different parameters used for stored procedure?

ASSIGNMENT NUMBER: A8

Revised On: 15/06/2018

TITLE	Database Trigger
PROBLEM STATEMENT /DEFINITION	• Write database trigger to keep track records on library table.
OBJECTIVE	• Understand the concept of database triggers. • Understand the Mysql commands
S/W PACKAGES AND HARDWARE APPARATUS USED	• MY SQL • PC with the configuration as Latest Version of 64 bit Operating Systems,Open Source Fedora-GHz. 8 G.B. RAM, 500 G.B. HDD, 15"Color Monitor, Keyboard, Mouse
REFERENCES	1. Silberschatz A., Korth H., Sudarshan S., "Database System Concepts", 5th Edition, McGraw Hill Publishers, 2002, ISBN 0-07-120413-X 2. Connally T., Begg C., "Database Systems", 3rd Edition, Pearson Education, 2002, ISBN 81-7808-861-4 3. mysql.com/docs/mysql-tutorial-excerpt-5.1-en.pdf
STEPS	**Refer to details**
INSTRUCTIONS FOR	• **Date** • **Title** • **Problem Definition** • **Learning Objectives**

WRITING JOURNAL	• **Learning Outcomes** • **Theory** • **Class Diagram/ER Diagram** • **Test cases** • **Program Listing** • **Output** • **Conclusion**

Problem statement:

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

Write a database trigger on Student 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 Alumni table.

Student (Rollno,Name,DateofAdmission,branch,percent,Status)

Aim: Study of database trigger.

Pre-requisite:

Basic knowledge of Mysql Commands and Mysql database

Learning Objectives:

- To understand Database trigger and its types.

Learning Outcomes:

The students will be able to

- Implement and apply types of database trigger.

Theory:

- **What are triggers?**

A trigger defines an action the database should take when some database-related event (such as inserts, updates, deletes) occurs. Triggers are similar to procedures, in that they are named PL/SQL blocks.

Differences between Procedures and Triggers

A procedure is executed explicitly from another block via a procedure call with passing arguments, while a trigger is executed (or fired) **implicitly** whenever the triggering event (**DML**: INSERT, UPDATE, or DELETE) happens, and a trigger doesn't accept arguments.

When triggers are used?

• Maintaining complex integrity constraints (referential integrity) or business rules
• Auditing information in a table by recording the changes.
• Automatically signaling other programs that action needs to take place when changes are made to a table
• Collecting/maintaining statistical data.

CREATE [OR REPLACE] TRIGGER trigger_name
{BEFORE | AFTER} {INSERT|UPDATE|DELETE} [OF column]
ON table_reference
[FOR EACH ROW [WHEN trigger_condition]]

[DECLARE] optional, for declaring local variables trigger_body;

- Note that the DECLARE keyword is back in Trigger.
- The Trigger_body is executed when an event (Insert, Update, Delete operation) occurs.

Trigger names

Triggers exist in a separate namespace from procedure, package, tables (that share the same namespace), which means that a trigger can have the same name as a table or procedure.

Types of triggers

There are two types of triggers in Oracle including row-level triggers and statement-level triggers

Row-level triggers for **data**-related activities

• Row-level triggers execute **once for each row** in a transaction.
• Row-level triggers are the most common type of triggers; they are often used in data auditing applications.

• Row-level trigger is identified by the **FOR EACH ROW** clause in the CREATE TRIGGER command.

Statement-level triggers for **transaction**-related activities

• Statement-level triggers execute **once for each transaction**. For example, if a single transaction inserted 500 rows into the Customer table, then a statement-level trigger on that table would only be executed once.
• Statement-level triggers therefore are not often used for *data-related* activities; they are normally used to enforce additional security measures on the types of transactions that may be performed on a table.
• Statement-level triggers are the default type of triggers created and are identified by **omitting** the **FOR EACH ROW** clause in the CREATE TRIGGER command.

Before and After Triggers

• Since triggers occur because of events, they may be set to occur immediately before or after those events. The events that execute triggers are database transactions, triggers can be executed immediately BEFORE or AFTER the statements INSERTs, UPDATEs, DELETEs.
• AFTER row-level triggers are frequently used in auditing applications, since they do not fire

until the row has been modified. Clearly, there is a great deal of flexibility in the design of a trigger.

Valid trigger types (possible combination of triggers)

• **Statement** (INSERT, DELETE, UPDATE), **Timing** (BEFORE, AFTER),
Level (Row-level, Statement-level)
The values for the statement, timing, and level determine the types of the triggers. There are total of 12 possible types of triggers: 3*2*2 = **12**

An Example of Trigger -- for the table major_stats (major, total_credits, total_students);

CREATE OR REPLACE TRIGGER updateMajorStats
AFTER INSERT **OR** DELETE **OR** UPDATE **ON** students -- Oracle will check the status of this table

DECLARE-- unlike a procedure, use DECLARE keywordCURSOR c_statistics IS
SELECT major, COUNT(*) total_students, SUM(current_credits) total_credits
FROM students
GROUP BY major;
BEGIN
FOR v_statsRecord IN c_statistics LOOP
UPDATE major_stats
SET total_credits = v_statsRecord.total_credits,
total_students = v_statsRecord.total_students
WHERE major = v_statsRecord.major;

IF SQL%NOTFOUND THEN
INSERT INTO major_stats(major, total_credits, total_students)
VALUES(v_statsRecord.major,
v_statsRecord.total_credits, v_statsRecord.total_students);
END IF;
END LOOP;
END updateMajorStats;

FAQ :

1. What is the difference between row level and Table level triggers?
2. What is the difference between BEFORE and AFTER in Database Triggers?
3. What is the advantage of stored procedure over the database triggers?
4. What are database triggers and how many types of triggers?
5. What is the difference between stored procedure and trigger?

Oral/Review:

1. What are the commands to enable and disable trigger?
2. How to check errors in trigger?
3. How many TRIGGERS are allowed in MySql table?
4. Which statement is used to create a trigger?
5. Triggers not supported for which opration?
6. Which statement is used to remove a trigger?
7. Before MySQL 5.1.6 which privilege was required to create and drop triggers?
8. Does Mysql enable the enforcement of the data integrity constraints?
9. Does TRIGGER privilege is used for the table?

ASSIGNMENT NUMBER: B1

Revised On: 15/06/2018

TITLE	Study of Open Source NOSQL Database: MongoDB (Installation, Basic CRUD operations, Execution)
PROBLEM STATEMENT /DEFINITION	Implement database with suitable example using Mongo DB and Implement Study of Open Source NOSQL Database: MongoDB (Installation, Basic CRUD operations, Execution)
OBJECTIVE	• Understand the concept of NOSQL DB. • Understand the concept of Mongo DB with CRUD operation • Understand the basic installation and administrative commands of Mongo DB .
S/W PACKAGES AND HARDWARE APPARATUS USED	• MongoDB • PC with the configuration as Latest Version of 64 bit Operating Systems,Open Source Fedora-GHz. 8 G.B. RAM, 500 G.B. HDD, 15"Color Monitor, Keyboard, Mouse
REFERENCES	• MongoDB: The Definitive Guide by Kristina Chodorow • http://docs.mongodb.org/manual/
STEPS	Refer to details
INSTRUCTIONS FOR WRITING JOURNAL	• **Title** • **Problem Definition** • **Learning Objectives** • **Theory** • **Class Diagram/ER Diagram** • **Test cases**

	• **Program Listing** • **Output** • **Conclusion**

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

Pre-requisite:
Basic knowledge SQL/NOSQL.

Learning Objectives:

- Understand the concept of NOSQL DB.
- Understand the concept of Mongo DB with CRUD operation
- Understand the basic installation and administrative commands of Mongo DB .
-

Learning Outcomes:

The students will be able to

- Implement the commands .
- Implement the Database in Mongo DB.

Theory:

MongoDB is an open-source document database that provides high performance, high availability, and automatic scaling.

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

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.

Document Database

A record in MongoDB is a document, which is a data structure composed of field and value pairs. MongoDB documents are similar to JSON objects. The values of fields may include other documents, arrays, and arrays of documents.

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.

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

Advantages of MongoDB over RDBMS

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

Why Use MongoDB?

1. Document Oriented Storage − Data is stored in the form of JSON style documents.
2. Index on any attribute
3. Replication and high availability
4. Auto-sharding
5. Rich queries
6. Fast in-place updates
7. Professional support by MongoDB

Where to Use MongoDB?

1. Big Data
2. Content Management and Delivery
3. Mobile and Social Infrastructure
4. User Data Management
5. Data Hub

FAQ :

1. What is difference between SQL vs NOSQL?
2. How does MongoDB provide consistency?
3. Give different types of NOSQL databases?
4. What is Aggregation in MongoDB?
5. What are CRUD operations?

Oral/Review:

1. Which syntax is used to create collection in mongodb?
2. Which command is used for inserting a document?
3. What is collection in monodb?

ASSIGNMENT NUMBER:B2

Revised On: 15/06/2018

TITLE	Design and Develop MongoDB Queries using CRUD operations.
PROBLEM STATEMENT /DEFINITION	Design and Develop MongoDB Queries using CRUD operations. (Use CRUD operations, SAVE method, logical operators)
OBJECTIVE	• To understand & implement the CRUD operations in Mongo DB.
S/W PACKAGES AND HARDWARE APPARATUS USED	• MongoDB • PC with the configuration as Latest Version of 64 bit Operating Systems,Open Source Fedora-GHz. 8 G.B. RAM, 500 G.B. HDD, 15"Color Monitor, Keyboard, Mouse
REFERENCES	• MongoDB: The Definitive Guide by Kristina Chodorow • http://docs.mongodb.org/manual/
STEPS	Refer to details
INSTRUCTIONS FOR WRITING JOURNAL	• Title • Problem Definition • Learning Objectives • Theory • Class Diagram/ER Diagram • Test cases • Program Listing • Output • Conclusion

Aim: Implement Mongo DB and Implement Basic operations

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

Pre-requisite:
Basic knowledge SQL/NOSQL.

Learning Objectives:

- To understand & implement the CRUD operations in Mongo DB

Learning Outcomes:

The students will be able to

- Implement the commands on two tier.
- Implement the Database in Mongo DB.

Theory:

Mongo DB:

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. 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.

Sample document
Below given 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 didn't provide then MongoDB provide 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.

Advantages of MongoDB over RDBMS

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

MongoDB CRUD operations with Python (Pymongo)

PyMongo is a Python distribution that contains tools for working with MongoDB, So in this blog post let's see some basic methods that perform CRUD operations to a collection.

Install pymongo:

pip install pymongo

Connecting to with Database with PyMongo:

5. To perform CRUD operations first need to establish the connection using Mongo client

```
>>> from pymongo import MongoClient
>>> client = MongoClient('localhost',27017)
```

6. Next step is to connect to the database (test)

```
db = client.test
```

7. Now retrieve the collection (person) from the database

```
col = db.person
```

Now we are ready to perform the actual CRUD operations.

CRUD Operations

C – Create : Mongo store the data in the form of JSON objects. So every record for a collection in mongo is called a **document**. If the collection does not currently exist, insert operations will create the collection. We can insert the documents into collection in 3 ways.

1. insert_one()
2. insert_many()
3. insert()

1. **insert_one():** insert_one() inserts a single document into a collection.

```
col.insert_one(
{
name: "John",
salary: 100 ,
}
)
```

2. **insert_many():** insert_many() inserts multiple documents into a collection.

```
col.insert_many(
[{ name: "Jeorge", salary: 100},
```

{ name: "Steve", salary: 100},

{ name: "David", salary: 100}]

)

3. **insert():** insert() can be used to insert single or array or documents.

```
# single document
col.insert({ name: "Jeorge", salary: 100})
# array of documents
col.insert([{ name: "Jeorge", salary: 100}, { name: "Steve", salary: 100}])
```

R - Read: We can retrieve the documents from a collection using 2 methods.

2. find()
3. find_one()

1. **find():** find() function will return with all the documents in that collection. By default it returns a cursor object.

```
col.find()
```

2. **find_one():** find_one() returns the first document in the collection.

```
col.find_one()
{u'salary': 100, u'_id': ObjectId('57611a711aa303032ad5ba9b'), u'name': u'John'}
```

D- Delete: We can delete the documents in the collection using the following methods.

3. delete_one()
4. delete_many()

Both these methods will return a **DeleteResult** object. The general syntax for the above methods is as follows.

<method_name>(condition)

Following are the examples how we use the delete_one() and delete_many() methods, both returns the **DeleteResult** object.

```
>>> col.delete_one({"name":"John"})
<pymongo.results.DeleteResult object at 0x7f8fbe7fba00>

>>> col.delete_many({"name":"John"})

<pymongo.results.DeleteResult object at 0x7f8fbe7fb960>
```

U- Update: We can update the documents from the collection with the following methods.

- update()
- update_one()
- update_many()
- replace_one()

The general syntax for all the above methods is

<method_name>(condition, update_or_replace_document, upsert=False, bypass_document_validation=False)

Here,
condition: A query that matches the document to replace.
update_or_replace_document: The new document.
upsert (optional): If True, perform an insert if no documents match the filter.
bypass_document_validation: (optional) If True, allows the write to opt-out of document level validation. Default is False.

 # update_one

```
>>> col.update_one({"name":"John"}, {"$set":{"name":"Joseph"}})
<pymongo.results.UpdateResult object at 0x7f8fbe7fb910>
```

 # update_many

```
>>> col.update_many({"name":"John"}, {"$set":{"name":"Joseph"}})
<pymongo.results.UpdateResult object at 0x7f8fbe7fb7d0>
```

 # update

```
>>> col.update({"name":"John"}, {"$set":{"name":"Jeorge"}}){'updatedExisting': False, u'nModified': 0, u'ok': 1, u'n': 0}
```

#replace_one

>>> col.replace_one({"name":"John"}, {"name":"Jeorge"}) <pymongo.results.UpdateResult object at 0x7f8fbe7fb910>

Logical Query Operators

- **$or -** Joins query clauses with a logical **OR** returns all documents that match the conditions of either clause.

{$or:[{<expression1>}, {<expression2>},...,{<expressionN>}]}

- **$and -** Joins query clauses with a logical **AND** returns all documents that match the conditions of both clauses.

{$and:[{<expression1>},{<expression2>},...,{<expressionN>}]}

- **$not -** Inverts the effect of a query expression and returns documents that do ***not*** match the query expression.

{ field: { $not: { <operator-expression> } } }

- **$nor -** Joins query clauses with a logical **NOR** returns all documents that fail to match both clauses.

{$nor:[{<expression1>},{<expression2>},...{<expressionN>}]}

FAQs:

1. Explain CRUD operations in MongoDB with suitable example?
2. Can we apply relationship in different collections, justify your answer?
3. Explain different applications of NOSQL?
4. Explain the MongoDB on Two tier architecture?
5. Explain basic commands of MongoDB ?

Oral/Review Questions:

1. Explain Index in MongoDb
2. What is difference between table in SQL and Collection in MONGODB?
3. What us use of Findone()?
4. How to create a user at Mongodb server?
5. How to insert the data in NOSQL environment?

ASSIGNMENT NUMBER: B3

Revised On: 15/06/2018

TITLE	Implement aggregation and indexing with suitable example using MongoDB
PROBLEM STATEMENT /DEFINITION	Implement aggregation and indexing with suitable example using MongoDB.
OBJECTIVE	• Understand indexing and aggregation concept on MongoDB
S/W PACKAGES AND HARDWARE APPARATUS USED	Mongodb Operating Systems • (64-Bit)64-BIT Fedora 17 or latest 64-BIT Update of Equivalent Open source OS or latest 64-BIT Version and update of Microsoft Windows 7 Operating System onwards • Programming Tools (64-Bit) Latest Open source update of Eclipse Programming frame work, MongoDB 2.6.
REFERENCES	**1.** MongoDB: The Definitive Guide, 2nd Edition,Powerful and Scalable Data Storage By Kristina Chodorow Publisher: O'Reilly Media **2.** http://docs.mongodb.org/manual
STEPS	Refer to details
INSTRUCTIONS FOR WRITING JOURNAL	• Title • Problem Definition • Learning Objectives • Theory

	• Class Diagram/ER Diagram • Test cases • Program Listing • Output • Conclusion

Aim: Implement aggregation and indexing with suitable example using MongoDB

Problem Statement: Implement aggregation and indexing with suitable example using MongoDB

Requirements:

- Computer System with Windows/Linux/Open Source Operating System.
- MongoDB Server

Learning Objectives

- To understand indexing and aggregation concept in MongoDB.

Theory:

Indexing:

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

Indexes are special data structures, which store a small portion of the data set in an easy to traverse form. The index stores the value of a specific field or set of fields, ordered by the value

of the field as specified in index. Indexing can be achieved on any field in a document using ensure Index () method.ensureIndex ()

Syntax:

>db.COLLECTION_NAME.ensureIndex

({KEY:1})

Here key is the name of filed on which you want to create index and 1 is for ascending order.

To create index in descending order one need to use -

1.Example:

>db.ensureIndex({eid:1})

This is simple/unique index. We can define index on multiple fields as well resulting into composite index.

>db.ensureIndex({eid:1,ename:-1}) Sole purpose of indexing lies in faster retrieval of data by organizing data in organized format.

Aggregation:

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.

In sql count(*) and with group by is an equivalent of MongoDB aggregation. The aggregate () Method For the aggregation in MongoDB one should use aggregate() method.

Syntax:

>db.COLLECTION_NAME.aggregate(AGGREGATE_OPERATION)

Aggregate operation could be finding sum on particular field/key or taking an average or finding maximum or minimum values associated with particular field from various documents in a single collection.

Example:

Assume a sample collection with sample documents inserted as below:

```
> db.emp1.insert(
.. [{eid:101,ename:"sachin",dept:"IT",sal:40000},
.. {eid:110,ename:"pranav",dept:"IT",sal:60000},
.. {eid:105,ename:"manoj",dept:"COMP",sal:45000},
.. {eid:102,ename:"yogesh",dept:"FE",sal:20000}])
```

If we want to display all employees based on their departments along with average salary of particular department we can write following query:

```
> db.emp1.aggregate([{$group:{_id:"$dept","avg
sal":{$avg:"$sal"}}}])
```

The above query computes average salary per department.

Following query displays number of employees associated along with particular department.

```
db.emp1.aggregate([{$group:{_id:"$dept","number of emp":{$sum:1}}}])
```

In the above example we have grouped documents by field dept and on each occurance of dept previous value of sum is incremented. The various available aggregation expressions are listed below:

Expression	Description
$sum	Sums up the defined value from all documents in the collection.
$avg	It computes average of defined value from all documents in the collection.
$max	It displays maximum of defined value from all documents in the collection.
$min	It displays minimum of defined value from all documents in the collection.

FAQs:

1. What is importance of Indexing?
2. Explain different types of indexing with suitable example?
3. Differentiate between SQL vs. MongoDB indexing with suitable example?
4. What is mean by aggregation? Explain aggregation in MongDB with suitable example?

Oral Questions

1. What changes you observe after performing Indexing?
2. How to remove specific document from MongoDB collection?
3. Execute all queries solved earlier on MySQL using MongoDB.
4. Can we specify more than one aggregate function simultaneously in MongoDB?
5. How to create a index in MongoDB?

ASSIGNMENT NUMBER: B4

Revised On: 15/06/2018

TITLE	Map reduce operation with suitable example using MongoDB
PROBLEM STATEMENT /DEFINITION	Write an example of Map reduce using MongoDB
OBJECTIVE	• To understand concept of Map-reduce as data processing paradigm for condensing large volumes of data into useful aggregated results.
S/W PACKAGES AND HARDWARE APPARATUS USED	Operating Systems (64-Bit)64-BIT Fedora 17 or latest 64-BIT Update of Equivalent Open source OS or latest 64-BIT Version and update of Microsoft Windows 7 Operating System onwards Programming Tools (64-Bit) Latest Open source update of Eclipse Programming frame work, TC++, GTK++, mongoDB 2.6.
REFERENCES	http://docs.mongodb.org/manual • MongoDB: The Definitive Guide, 2nd Edition,Powerful and Scalable Data Storage By Kristina Chodorow Publisher: O'Reilly Media
STEPS	Refer to details
INSTRUCTIONS FOR WRITING JOURNAL	1. Title

	2. Problem Definition 3. Learning Objectives 4. Theory 5. Class Diagram/ER Diagram 6. Test cases 7. Program Listing 8. Output 9. Conclusion

THEORY:

Map Reduce

- MapReduce is a powerful and flexible tool for aggregating data.
- It can solve some problems that are too complex to express using the aggregation framework's query language.
- It uses JavaScript as its "query language".
- It is fairly slow and should not be used for real-time data analysis.
- MapReduce can be easily parallelized across multiple servers.

Map Phase

- Map an operation onto every document in a collection. That operation could be either “do nothing” or “emit these keys with X values.”

Intermediate stage

- Keys are grouped and lists of emitted values are created for each key.

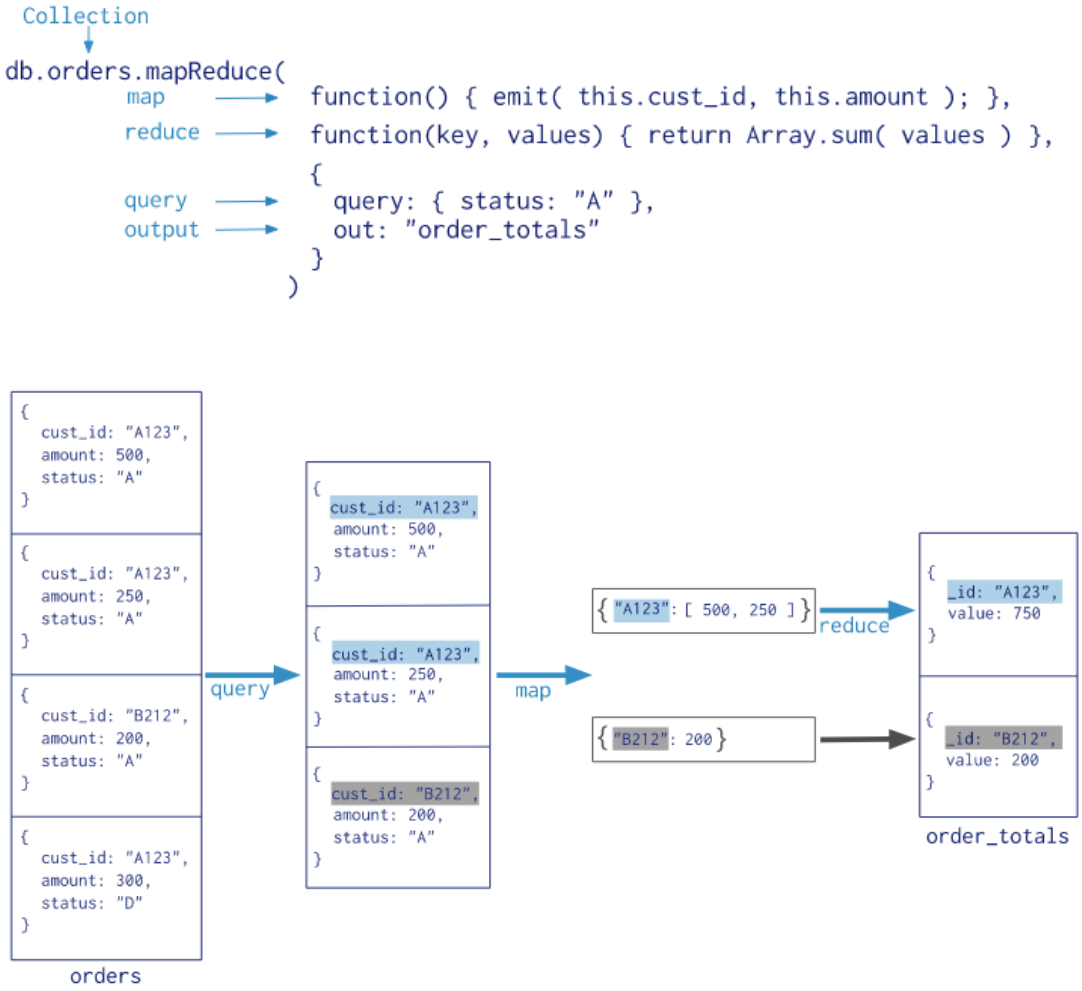
Reduce phase

- Takes this list of values and reduces it to a single element.
- This element is returned to the shuffle step until each key has a list containing a single value: the result.

Syntax:

>db.collection.mapReduce(

```
   function() {emit(key,value);},  //map function
   function(key,values) {return reduceFunction}, {   //reduce function
      out: collection,
      query: document,
      sort: document,
      limit: number
   }
)
```



In this map-reduce operation, MongoDB applies the map phase to each input document (i.e. the documents in the collection that match the query condition). The map function emits key-value pairs. For those keys that have multiple values, MongoDB applies the reduce phase, which collects and condenses the aggregated data. MongoDB then stores the results in a collection. Optionally, the output of the reduce function may pass through a finalize function to further condense or process the results of the aggregation.

All map-reduce functions in MongoDB are JavaScript and run within the mongod process. Map-reduce operations take the documents of a single collection as the input and can perform any arbitrary sorting and limiting before beginning the map stage. mapReduce can return the results of a map-reduce operation as a document, or may write the results to collections. The input and the output collections may be sharded.

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

In the above syntax −

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

Using MapReduce

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

```
{
  "post_text": "PICT",
  "user_name": "mark",
  "status":"active"
}
```

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

```
>db.posts.mapReduce(
  function() { emit(this.user_id,1); },

  function(key, values) {return Array.sum(values)}, {
    query:{status:"active"},
```

```
    out:"post_total"
  }
)
```

The above mapReduce query outputs the following result –

```
{
  "result" : "post_total",
  "timeMillis" : 9,
  "counts" : {
    "input" : 4,
    "emit" : 4,
    "reduce" : 2,
    "output" : 2
  },
  "ok" : 1,
}
```

The result shows that a total of 4 documents matched the query (status:"active"), the map function emitted 4 documents with key-value pairs and finally the reduce function grouped mapped documents having the same keys into 2.

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

```
>db.posts.mapReduce(
  function() { emit(this.user_id,1); },
  function(key, values) {return Array.sum(values)}, {
    query:{status:"active"},
    out:"post_total"
  }

).find()
```

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

```
{ "_id" : "tom", "value" : 2 }
{ "_id" : "mark", "value" : 2 }
```

In a similar manner, MapReduce queries can be used to construct large complex aggregation queries. The use of custom Javascript functions make use of MapReduce which is very flexible and powerful.

FAQs:

1. What is the need of MapReduce?
2. Explain Map reduce framework with suitable example?
3. What is emit function?
4. How map reduce work in MongoDB?

Oral Questions

1. What is mean by Map and reduce?
2. Is there any similarity between Map reduce in MongoDB and Mapreduce in Hadoop?
3. What is difference between Map reduce and aggregation ?

ASSIGNMENT NUMBER: B5

Revised On: 15/06/2018

TITLE	Implement 5 Basic query using Mongo DB
PROBLEM STATEMENT /DEFINITION	Design and Implement any 5 query using MongoDB
OBJECTIVE	1. Understand the concept of Mongo DB. 2. Understand the concept of Mongo DB on Two tier 3. Understand the basic commands of Mongo DB .
S/W PACKAGES AND HARDWARE APPARATUS USED	• Mongo DB • PC with the configuration as Latest Version of 64 bit Operating Systems, Open Source Fedora-GHz. 8 G.B. RAM, 500 G.B. HDD, 15"Color Monitor, Keyboard, Mouse
REFERENCES	1. MongoDB: The Definitive Guide by Kristina Chodorow 2. http://docs.mongodb.org/manual/
STEPS	Refer to details
INSTRUCTIONS FOR WRITING JOURNAL	• **Title** • **Problem Definition** • **Learning Objectives** • **Theory** • **Class Diagram/ER Diagram** • **Test cases** • **Program Listing** • **Output**

	• **Conclusion**

Aim: Design and Implement any 5 query using MongoDB

Pre-requisite:
Basic knowledge SQL/NOSQL.

Learning Objectives:

4. To understand & implement the various commands of Mongo DB.

Learning Outcomes:

The students will be able to

3. Implement the commands on two tier.
4. Implement the Database in Mongo DB.

Theory:

Mongo DB:

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. 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.

Sample document

Below given 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 didn't provide then MongoDB provide 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.

- **Advantages of MongoDB over RDBMS**

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

5. **MongoDB Create Database**

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

Example:
If you want to create a database with name <mydb>, then use DATABASE statement would be as follows:
>use mydb
switched to db mydb

To check your currently selected database use the command db
>db
>Mydb

Below given table shows the relationship of RDBMS terminology with MongoDB

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

Sample document

Below given 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:newDate(2011,1,20,2,15),
like:0
},
{
user:'user2',
message:'My second comments',
dateCreated:newDate(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 didn't provide then MongoDB provide 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.

Key Features

1. High Performance
   - MongoDB provides high performance data persistence. In particular,
   - Support for embedded data models reduces I/O activity on database system.
   - Indexes support faster queries and can include keys from embedded documents and arrays.
2. High Availability
   - To provide high availability, MongoDB's replication facility, called replica sets, provide:
   - Automatic failover.
   - Data redundancy.
   - A replica setis a group of MongoDB servers that maintain the same data set, providing redundancy and
   - Increasing data availability.
3. Automatic Scaling
   - MongoDB provides horizontal scalability as part of its core functionality.
   - Automatic sharding distributes data across a cluster of machines.
   - Replica sets can provide eventually-consistent reads for low-latency high throughput deployments.

Why should use MongoDB

- Document Oriented Storage : Data is stored in the form of JSON style documents
- Index on any attribute
- Replication & High Availability
- Auto-Sharding
- Rich Queries
- Fast In-Place Updates
- Professional Support By MongoDB

Where should use MongoDB?

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

- Data Hub

MongoDB Create Database

The use Command

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

Syntax:

Basic syntax of **use DATABASE** statement is as follows:

```
>use DATABASE_NAME
```

Example:

If you want to create a database with name **<mydb>**, then **use DATABASE** statement would be as follows:

```
>usemydb
switched to dbmydb
```

To check your currently selected database use the command **db**

```
>db
Mydb
```

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

```
>show dbs
local0.78125GB
test0.23012GB
```

Your created database (mydb) is not present in list. To display database you need to insert atleast one document into it.

```
>db.movie.insert({"name":"tutorials point"})
>show dbs
local0.78125GB
mydb0.23012GB
test0.23012GB
```

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

MongoDB Drop Database

The dropDatabase() Method

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

Syntax:

Basic syntax of **dropDatabase()** command is as follows:

```
>db.dropDatabase()
```

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

Example:

First, check the list available databases by using the command **show dbs**

```
>show dbs
local0.78125GB
mydb0.23012GB
test0.23012GB
>
```

If you want to delete new database **<mydb>**, then **dropDatabase()** command would be as follows:

```
>usemydb
switched to dbmydb
>db.dropDatabase()
>{"dropped":"mydb","ok":1}
>
```

Now check list of databases

```
>show dbs
local0.78125GB
test0.23012GB
>
```

MongoDB Create Collection

The createCollection() Method

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

Syntax:

Basic syntax of **createCollection()** command is as follows
db.createCollection(name, options)
In the command, **name** is name of collection to be created. An **option is** a document and used to specify configuration of collection

Parameter	Type	Description
Name	String	Name of the collection to be created
Options	Document	(Optional) Specify options about memory size and indexing

Options parameter is optional, so you need to specify only name of the collection. Following is the list of options you can use:

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

While inserting the document, MongoDB first checks size field of capped collection, then it checks max field.

Examples:

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

```
>use test
switched to db test
```

```
>db.createCollection("mycollection")
{"ok":1}
>
```

You can check the created collection by using the command **show collections**

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

Following example shows the syntax of **createCollection()** method with few important options:

```
>db.createCollection("mycol",{ capped :true,autoIndexID:true, size :6142800, max :10000})
{"ok":1}
>
```

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

```
>db.tutorialspoint.insert({"name":"tutorialspoint"})
>show collections
mycol
mycollection
system.indexes
tutorialspoint
>
```

MongoDB Drop Collection

The drop() Method

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

Syntax:

Basic syntax of **drop()** command is as follows

```
>db.COLLECTION_NAME.drop()
```

Example:

First, check the available collections into your database **mydb**

```
>use mydb
switched to dbmydb
```

```
>show collections
mycol
mycollection
system.indexes
tutorialspoint
>
```

Now drop the collection with the name **mycollection**

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

Again check the list of collections into database

```
>show collections
mycol
system.indexes
tutorialspoint
>
```

drop() method will return true, if the selected collection is dropped successfully otherwise it will return false

MongoDB Datatypes

MongoDB supports many datatypes whose list is given below:

- **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**: 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

MongoDB - Insert Document

The insert() Method

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

Syntax

Basic syntax of **insert()** command is as follows:

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

Example

```
>db.mycol.insert({
  _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
})
```

Here **mycol** is our collection name, as created in previous tutorial. If the collection doesn't exist in the database, then MongoDB will create this collection and then insert document into it.

In the inserted document if we don't specify the _id parameter, then MongoDB assigns an unique ObjectId for this document.

_id is 12 bytes hexadecimal number unique for every document in a collection. 12 bytes are divided as follows:

```
_id: ObjectId(4 bytes timestamp, 3 bytes machine id, 2 bytes process id, 3 bytes incrementer)
```

To insert multiple documents in single query, you can pass an array of documents in insert() command.

Example

```
>db.post.insert([
```

```
{
title: 'MongoDB Overview',
description: 'MongoDB is no sql database',
by: 'tutorials point',
url: 'http://www.tutorialspoint.com',
tags: ['mongodb', 'database', 'NoSQL'],
likes: 100
},
{
title: 'NoSQL Database',
description: 'NoSQL database doesn't have tables',
by: 'tutorials point',
url: 'http://www.tutorialspoint.com',
tags: ['mongodb', 'database', 'NoSQL'],
likes: 20,
comments: [
     {
user:'user1',
message: 'My first comment',
dateCreated: new Date(2013,11,10,2,35),
like: 0
     }
  ]
}
])
```

To insert the document you can use **db.post.save(document)** also. If you don't specify **_id** in the document then **save()** method will work same as **insert()** method. If you specify _id then it will replace whole data of document containing _id as specified in save() method.

MongoDB - Query Document

The find() Method

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

Syntax

Basic syntax of **find()** method is as follows

```
>db.COLLECTION_NAME.find()
```

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

The pretty() Method

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

Syntax:

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

Example

```
>db.mycol.find().pretty()
{
  "_id": ObjectId(7df78ad8902c),
  "title": "MongoDB Overview",
  "description": "MongoDB is no sql database",
  "by": "tutorials point",
  "url": "http://www.tutorialspoint.com",
  "tags": ["mongodb", "database", "NoSQL"],
  "likes": "100"
}
>
```

Apart from find() method there is **findOne()** method, that reruns only one document.

RDBMS Where Clause Equivalents in MongoDB

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

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

AND in MongoDB

Syntax:

In the **find()** method if you pass multiple keys by separating them by ',' then MongoDB treats it **AND** condition. Basic syntax of **AND** is shown below:

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

Example

Below given example will show all the tutorials written by 'tutorials point' and whose title is 'MongoDB Overview'

```
>db.mycol.find({"by":"tutorialspoint","title": "MongoDB Overview"}).pretty()
{
  "_id": ObjectId(7df78ad8902c),
  "title": "MongoDB Overview",
  "description": "MongoDB is no sql database",
  "by": "tutorials point",
  "url": "http://www.tutorialspoint.com",
  "tags": ["mongodb", "database", "NoSQL"],
  "likes": "100"
}
>
```

For the above given example equivalent where clause will be **' where by='tutorials point' AND title='MongoDB Overview' '**. You can pass any number of key, value pairs in find clause.

OR in MongoDB

Syntax:

To query documents based on the OR condition, you need to use **$or** keyword. Basic syntax of **OR** is shown below:

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

```
   }
).pretty()
```

Example

Below given example will show all the tutorials written by 'tutorials point' or whose title is 'MongoDB Overview'

```
>db.mycol.find({$or:[{"by":"tutorials point"},{"title": "MongoDB Overview"}]}).pretty()
{
   "_id": ObjectId(7df78ad8902c),
   "title": "MongoDB Overview",
   "description": "MongoDB is no sql database",
   "by": "tutorials point",
   "url": "http://www.tutorialspoint.com",
   "tags": ["mongodb", "database", "NoSQL"],
   "likes": "100"
}
>
```

Using AND and OR together

Example

Below given example will show the documents that have likes greater than 100 and whose title is either 'MongoDB Overview' or by is 'tutorials point'. Equivalent sql where clause is **'where likes>10 AND (by = 'tutorials point' OR title = 'MongoDB Overview')'**

```
>db.mycol.find("likes": {$gt:10}, $or: [{"by": "tutorials point"}, {"title": "MongoDB Overview"}]
}).pretty()
{
   "_id": ObjectId(7df78ad8902c),
   "title": "MongoDB Overview",
   "description": "MongoDB is no sql database",
   "by": "tutorials point",
   "url": "http://www.tutorialspoint.com",
   "tags": ["mongodb", "database", "NoSQL"],
   "likes": "100"
}
>
```

MongoDB Update Document

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

MongoDB Update() method

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

Syntax:

Basic syntax of **update()** method is as follows

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

Example

Consider the mycolcollectioin has following data.

```
{"_id":ObjectId(5983548781331adf45ec5),"title":"MongoDB Overview"}
{"_id":ObjectId(5983548781331adf45ec6),"title":"NoSQL Overview"}
{"_id":ObjectId(5983548781331adf45ec7),"title":"Tutorials Point Overview"}
```

Following example will set the new title 'New MongoDB Tutorial' of the documents whose title is 'MongoDB Overview'

```
>db.mycol.update({'title':'MongoDB Overview'},{$set:{'title':'New MongoDB Tutorial'}})
>db.mycol.find()
{"_id":ObjectId(5983548781331adf45ec5),"title":"New MongoDB Tutorial"}
{"_id":ObjectId(5983548781331adf45ec6),"title":"NoSQL Overview"}
{"_id":ObjectId(5983548781331adf45ec7),"title":"Tutorials Point Overview"}
>
```

By default mongodb will update only single document, to update multiple you need to set a paramter 'multi' to true.

```
>db.mycol.update({'title':'MongoDB Overview'},{$set:{'title':'New MongoDB Tutorial'}},{multi:true})
```

MongoDB Save() Method

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

Syntax

Basic syntax of mongodb**save()** method is shown below:

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

Example

Following example will replace the document with the _id '5983548781331adf45ec7'

```
>db.mycol.save(
{
"_id":ObjectId(5983548781331adf45ec7),"title":"Tutorials Point New Topic","by":"Tutorials Point"
}
)
>db.mycol.find()
{"_id":ObjectId(5983548781331adf45ec5),"title":"Tutorials Point New Topic","by":"Tutorials Point"}
{"_id":ObjectId(5983548781331adf45ec6),"title":"NoSQL Overview"}
{"_id":ObjectId(5983548781331adf45ec7),"title":"Tutorials Point Overview"}
>
```

MongoDB Delete Document

The remove() Method

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

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

Syntax:

Basic syntax of **remove()** method is as follows

```
>db.COLLECTION_NAME.remove(DELLETION_CRITTERIA)
```

Example

Consider the mycolcollectioin has following data.

```
{"_id":ObjectId(5983548781331adf45ec5),"title":"MongoDB Overview"}
{"_id":ObjectId(5983548781331adf45ec6),"title":"NoSQL Overview"}
{"_id":ObjectId(5983548781331adf45ec7),"title":"Tutorials Point Overview"}
```

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

```
>db.mycol.remove({'title':'MongoDB Overview'})
>db.mycol.find()
{"_id":ObjectId(5983548781331adf45ec6),"title":"NoSQL Overview"}
{"_id":ObjectId(5983548781331adf45ec7),"title":"Tutorials Point Overview"}
>
```

Remove only one

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

```
>db.COLLECTION_NAME.remove(DELETION_CRITERIA,1)
```

Remove all documents

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

```
>db.mycol.remove()
>db.mycol.find()
>
```

MongoDB Projection

In mongodb projection meaning is selecting only necessary data rather than selecting whole of the data of a document. If a document has 5 fields and you need to show only 3, then select only 3 fields from them.

The find() Method

MongoDB's **find()** method, explained in MongoDB Query Document accepts second optional parameter that is list of fields that you want to retrieve. In MongoDB when you execute **find()** method, then it displays all fields of a document. To limit this you need to set list of fields with value 1 or 0. 1 is used to show the filed while 0 is used to hide the field.

Syntax:

Basic syntax of **find()** method with projection is as follows

```
>db.COLLECTION_NAME.find({},{KEY:1})
```

Example

Consider the collection myycol has the following data

```
{"_id":ObjectId(5983548781331adf45ec5),"title":"MongoDB Overview"}
{"_id":ObjectId(5983548781331adf45ec6),"title":"NoSQL Overview"}
{"_id":ObjectId(5983548781331adf45ec7),"title":"Tutorials Point Overview"}
```

Following example will display the title of the document while quering the document.

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

{"title":"MongoDB Overview"}
{"title":"NoSQL Overview"}
{"title":"Tutorials Point Overview"}
>

Please note **_id** field is always displayed while executing **find()** method, if you don't want this field, then you need to set it as 0

MongoDB Limit Records

The Limit() Method

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

Syntax:

Basic syntax of **limit()** method is as follows
>db.COLLECTION_NAME.find().limit(NUMBER)

Example

Consider the collection myycol has the following data
{"_id":ObjectId(5983548781331adf45ec5),"title":"MongoDB Overview"}
{"_id":ObjectId(5983548781331adf45ec6),"title":"NoSQL Overview"}
{"_id":ObjectId(5983548781331adf45ec7),"title":"Tutorials Point Overview"}
Following example will display only 2 documents while quering the document.
>db.mycol.find({},{"title":1,_id:0}).limit(2)
{"title":"MongoDB Overview"}
{"title":"NoSQL Overview"}
>
If you don't specify number argument in **limit()** method then it will display all documents from the collection.

MongoDB Skip() Method

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

Basic syntax of **skip()** method is as follows
>db.COLLECTION_NAME.find().limit(NUMBER).skip(NUMBER)

Example:

Following example will only display only second document.
>db.mycol.find({},{"title":1,_id:0}).limit(1).skip(1)
{"title":"NoSQL Overview"}
>
Please note default value in **skip()** method is 0

MongoDB Sort Documents

The sort() Method

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

Syntax:

Basic syntax of **sort()** method is as follows

>db.COLLECTION_NAME.find().sort({KEY:1})

Example

Consider the collection myycol has the following data
{"_id":ObjectId(5983548781331adf45ec5),"title":"MongoDB Overview"}
{"_id":ObjectId(5983548781331adf45ec6),"title":"NoSQL Overview"}
{"_id":ObjectId(5983548781331adf45ec7),"title":"Tutorials Point Overview"}
Following example will display the documents sorted by title in descending order.
>db.mycol.find({},{"title":1,_id:0}).sort({"title":-1})
{"title":"Tutorials Point Overview"}
{"title":"NoSQL Overview"}
{"title":"MongoDB Overview"}
>

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

MongoDB Relationships

Relationships in MongoDB represent how various documents are logically related to each other. Relationships can be modeled via **Embedded** and **Referenced** approaches. Such relationships can be either 1:1, 1: N, N: 1 or N: N.
Let us consider the case of storing addresses for users. So, one user can have multiple addresses making this a 1: N relationship.

Following is the sample document structure of **user** document:


```
{
"_id":ObjectId("52ffc33cd85242f436000001"),
"name":"Tom Hanks",
"contact":"987654321",
"dob":"01-01-1991"
}
```


Following is the sample document structure of **address** document:


```
{
"_id":ObjectId("52ffc4a5d85242602e000000"),
"building":"22 A, Indiana Apt",
"pincode":123456,
"city":"Los Angeles",
"state":"California"
}
```


Modeling Embedded Relationships

In the embedded approach, we will embed the address document inside the user document.


```
{
"_id":ObjectId("52ffc33cd85242f436000001"),
"contact":"987654321",
"dob":"01-01-1991",
"name":"Tom Benzamin",
"address":[
{
"building":"22 A, Indiana Apt",
"pincode":123456,
"city":"Los Angeles",
"state":"California"
},
{
"building":"170 A, Acropolis Apt",
"pincode":456789,
"city":"Chicago",
"state":"Illinois"
}]
}
```

This approach maintains all the related data in a single document which makes it easy to retrieve and maintain. The whole document can be retrieved in a single query like this:

>db.users.findOne({"name":"TomBenzamin"},{"address":1})
Note that in the above query, **db** and **users** are the database and collection respectively.
The drawback is that if the embedded document keeps on growing too much in size, it can impact the read/write performance.

Modeling Referenced Relationships

This is the approach of designing normalized relationship. In this approach, both the user and address documents will be maintained separately but the user document will contain a field that will reference the address document's **id** field.

```
{
"_id":ObjectId("52ffc33cd85242f436000001"),
"contact":"987654321",
"dob":"01-01-1991",
"name":"Tom Benzamin",
"address_ids":[
ObjectId("52ffc4a5d85242602e000000"),
ObjectId("52ffc4a5d85242602e000001")
]
}
```

As shown above, the user document contains the array field **address_ids**which contains ObjectIds of corresponding addresses.Using these ObjectIds, we can query the address documents and get address details from there. With this approach, we will need two queries: first to fetch the **address_ids** fields from **user** document and second to fetch these addresses from **address** collection.

```
>var result =db.users.findOne({"name":"TomBenzamin"},{"address_ids":1})
>var addresses =db.address.find({"_id":{"$in":result["address_ids"]}})
```

Exercises:

1. create a database college first within that create a collection as Teacher information,which contain the information or fieldssuch as Teacher_id, name of a teacher, department of a teacher, salary and status of a teacher. Here status is whether teacher is approved by the university or not.

Implement the DDL & DML queries on the Teacher information collection and difference between SQL Commands and MongoDB commands.

FAQs:

1.Explain CRUD operations in MongoDB?

2.Explain data types in MongoDB?

3.Describe MongoDB programming environment with suitable example?

4.Explain the MongoDB on Two tier architecture?
5.Explain basic commands of MongoDB ?

Oral Questions

1. What is syntax for creating collection in MongoDB?
2. What is difference between find() and Findone()?
3. What is primary key in MondoDB?
4. Differentiate between CRUD operation MysQL and MongoDB?

ASSIGNMENT NUMBER:B6

Revised On: 15/06/2018

TITLE	Create simple objects and array objects using JSON
PROBLEM STATEMENT /DEFINITION	Create simple objects and array objects using JSON
OBJECTIVE	• Understand the concept objects in JSON • Understand the simple and array objects
S/W PACKAGES AND HARDWARE APPARATUS USED	PC with the configuration as Latest Version of 64 bit Operating Systems,Open Source Fedora-GHz. 8 G.B. RAM, 500 G.B. HDD, 15"Color Monitor, Keyboard, Mouse
REFERENCES	1.Silberschatz A., Korth H., Sudarshan S., "Database System Concepts", 5th Edition, McGraw Hill Publishers, 2002, ISBN 0-07-120413-X 2. Connally T., Begg C., "Database Systems", 3rd Edition, Pearson Education, 2002, ISBN 81-7808-861-4
STEPS	Refer to details
INSTRUCTIONS FOR WRITING JOURNAL	• Title • Problem Definition • Learning Objectives • Theory • Class Diagram/ER Diagram • Test cases

	• Program Listing • Output • Conclusion

Aim: Create simple objects and array objects using JSON

Problem Statement: Create simple objects and array objects using JSON

Pre-requisite:

Basic knowledge of HTML and JSON.

Learning Objectives:

- To understand the concept of JSON and creating array objects.

Theory:

JSON objects can be created with JavaScript. Let us see the various ways of creating JSON objects using JavaScript −

- Creation of an empty Object −

```
var JSONObj = {};
```

- Creation of a new Object −

```
var JSONObj = new Object();
```

- Creation of an object with attribute **bookname** with value in string, attribute **price** with numeric value. Attribute is accessed by using '.' Operator −

```
var JSONObj = { "bookname ":"VB BLACK BOOK", "price":500 };
```

```
<html>
   <head>
      <title>Creating Object JSON with JavaScript</title>

      <script language = "javascript" >

         var JSONObj = { "name" : "tutorialspoint.com", "year"  : 2005 };

         document.write("<h1>JSON with JavaScript example</h1>");
         document.write("<br>");
         document.write("<h3>Website Name = "+JSONObj.name+"</h3>");
         document.write("<h3>Year = "+JSONObj.year+"</h3>");

      </script>

   </head>

   <body>
   </body>

</html>
```

Creating Array Objects

The following example shows creation of an array object in javascript using JSON, save the below code as **json_array_object.htm** −

```
<html>
  <head>
    <title>Creation of array object in javascript using JSON</title>

    <script language = "javascript" >

      document.writeln("<h2>JSON array object</h2>");
```

```
var books = { "Pascal" : [
  { "Name"  : "Pascal Made Simple", "price" : 700 },
  { "Name"  : "Guide to Pascal", "price" : 400 }],

  "Scala"  : [
    { "Name"  : "Scala for the Impatient", "price" : 1000 },
    { "Name"  : "Scala in Depth", "price" : 1300 }]
}

var i = 0
document.writeln("<table border = '2'><tr>");

for(i = 0;i<books.Pascal.length;i++){
  document.writeln("<td>");
  document.writeln("<table border = '1' width = 100 >");
  document.writeln("<tr><td><b>Name</b></td><td width = 50>" +
books.Pascal[i].Name+"</td></tr>");
  document.writeln("<tr><td><b>Price</b></td><td width = 50>" + books.Pascal[i].price
+"</td></tr>");
  document.writeln("</table>");
  document.writeln("</td>");
}

for(i = 0;i<books.Scala.length;i++){
  document.writeln("<td>");
  document.writeln("<table border = '1' width = 100 >");
  document.writeln("<tr><td><b>Name</b></td><td width = 50>" +
books.Scala[i].Name+"</td></tr>");
  document.writeln("<tr><td><b>Price</b></td><td width = 50>" +
books.Scala[i].price+"</td></tr>");
  document.writeln("</table>");
  document.writeln("</td>");
}

document.writeln("</tr></table>");
```

```
    </script>

  </head>

  <body>
  </body>

</html>
```

JSON or JavaScript Object Notation (JSON) is a lightweight is a text-based open standard designed for human-readable data interchange.
The JSON format was originally specified by Douglas Crockford, and is described in RFC 4627. The official Internet media type for JSON is application/json. The JSON filename extension is .json.
JSON or JavaScript Object Notation is a lightweight text-based open standard designed for human-readable data interchange. Conventions used by JSON are known to programmers which include C, C++, Java, Python, Perl etc.

- JSON stands for JavaScript Object Notation.
- This format was specified by Douglas Crockford.
- This was designed for human-readable data interchange
- It has been extended from the JavaScript scripting language.
- The filename extension is .json
- JSON Internet Media type is application/json
- The Uniform Type Identifier is public.json

Uses of JSON

- It is used when writing JavaScript based application which includes browser extension and websites.
- JSON format is used for serializing & transmitting structured data over network connection.
- This is primarily used to transmit data between server and web application.
- Web Services and API.s use JSON format to provide public data.
- It can be used with modern programming languages.

Characteristics of JSON

- Easy to read and write JSON.
- Lightweight text based interchange format

- Language independent.

Simple Example in JSON

Example shows Books information stored using JSON considering language of books and there editions:

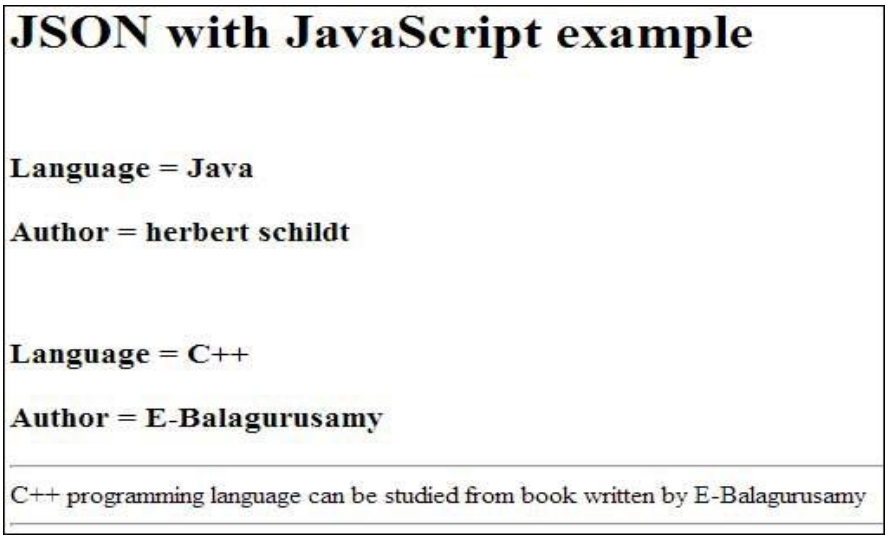
```
{
   "book": [
   {
     "id":"01",
     "language": "Java",
     "edition": "third",
     "author": "Herbert Schildt"
   },
   {
     "id":"07",
     "language": "C++",
     "edition": "second"
     "author": "E.Balagurusamy"
   }]
}
```

After understanding the above program we will try another example, let's save the below code as json.htm:

```
<html>
<head>
<title>JSON example</title>
<script language="javascript" >

  var object1 = { "language" : "Java", "author"  : "herbert schildt" };
  document.write("<h1>JSON with JavaScript example</h1>");
  document.write("<br>");
  document.write("<h3>Language = " + object1.language+"</h3>");
  document.write("<h3>Author = " + object1.author+"</h3>");

  var object2 = { "language" : "C++", "author"  : "E-Balagurusamy" };
  document.write("<br>");
  document.write("<h3>Language = " + object2.language+"</h3>");
  document.write("<h3>Author = " + object2.author+"</h3>");

  document.write("<hr />");
```

```
  document.write(object2.language + " programming language can be studied " +
  "from book written by " + object2.author);
  document.write("<hr />");

</script>
</head>
<body>
</body>
</html>
```

Now let's try to open json.htm using IE or any other javascript enabled browser, this produces the following result:



❖ **JSON - Syntax**

Let's have a quick look on JSON basic syntax. JSON syntax is basically considered as subset of JavaScript syntax, it includes the following:

Data is represented in name/value pairs

Curly braces hold objects and each name is followed by ':'(colon), the name/value pairs are separated by , (comma).

Square brackets hold arrays and values are separated by ,(comma).

Below is a simple example:

```
{
   "book": [
   {
      "id":"01",
      "language": "Java",
      "edition": "third",
      "author": "Herbert Schildt"
   },
   {
      "id":"07",
      "language": "C++",
      "edition": "second"
      "author": "E.Balagurusamy"
   }]
}
```

JSON supports following two data structures:

- Collection of name/value pairs: This Data Structure is supported by different programming language.
- Ordered list of values: It includes array, list, vector or sequence etc.

❖ **JSON – Data Types**

There are following datatypes supported by JSON format:

Type	Description
Number	double- precision floating-point format in JavaScript
String	double-quoted Unicode with backslash escaping
Boolean	true or false
Array	an ordered sequence of values
Value	it can be a string, a number, true or false, null etc
Object	an unordered collection of key:value pairs
null	Empty

1. Number

- It is a double precision floating-point format in JavaScript and it depends on implementation.
- Octal and hexadecimal formats are not used.
- No NaN or Infinity is used in Number.

The following table shows number types:

Type	Description
Integer	Digits 1-9, 0 and positive or negative
Fraction	Fractions like .3, .9
Exponent	Exponent like e, e+, e-,E, E+, E-

Syntax:

var json-object-name = { string : number_value, ...}

Example:

Example showing Number Datatype, value should not be quoted:
var obj = {marks: 97}

2. String

- It is a sequence of zero or more double quoted Unicode characters with backslash escaping.
- Character is a single character string i.e. a string with length 1.

The table shows string types:

Type	Description
"	double quotation
\	reverse solidus
/	solidus
b	backspace

Syntax:

var json-object-name = { string : "string value", ...}

Example:

Example showing String Datatype:

var obj = {name: 'Amit'}

3. Boolean

It includes true or false values.

Syntax:

var json-object-name = { string : true/false, ...}

Example:

var obj = {name: 'Amit', marks: 97, distinction: true}

4. Array

- It is an ordered collection of values.
- These are enclosed square brackets which means that array begins with .[. and ends with .]..
- The values are separated by ,(comma).
- Array indexing can be started at 0 or 1.
- Arrays should be used when the key names are sequential integers.

Syntax:

[value, ...]

Example:

Example showing array containing multiple objects:

```
{
  "books": [
   { "language":"Java" , "edition":"second" },
   { "language":"C++" , "lastName":"fifth" },
   { "language":"C" , "lastName":"third" }
  ]}
```

5. Object

- It is an unordered set of name/value pairs.
- Object are enclosed in curly braces that is it starts with '{' and ends with '}'.
- Each name is followed by ':'(colon) and the name/value pairs are separated by , (comma).
- The keys must be strings and should be different from each other.
- Objects should be used when the key names are arbitrary strings
-

Syntax:

{ string : value, ...}

Example:

Example showing Object:

```
{
 "id": "011A",
 "language": "JAVA",
 "price": 500,
}
```

6. Whitespace

It can be inserted between any pair of tokens. It can be added to make code more readable. Example shows declaration with and without whitespace:

Syntax:

{string:" ",....}

Example:

```
var i= "   sachin";
var j = "  saurav"
```

7. null

It means empty type.

Syntax:

null

Example:

```
var i = null;
```

```
if(i==1)
{
  document.write("<h1>value is 1</h1>");
}
else
{
  document.write("<h1>value is null</h1>");
}
```

8. JSON Value

It includes:

- number (integer or floating point)
- string
- boolean
- array
- object
- null

Syntax:

String | Number | Object | Array | TRUE | FALSE | NULL

Example:

```
var i =1;
var j = "sachin";
var k = null;
```

❖ **JSON - Objects**

➢ **Creating Simple Objects**

JSON objects can be created with Javascript. Let us see various ways of creating JSON objects using Javascript:

1. Creation of an empty Object:

```
var JSONObj = {};
```

2. Creation of new Object:

```
var JSONObj = new Object();
```

3. Creation of an object with attribute **bookname** with value in string, attribute **price** with numeric value. Attributes is accessed by using '.' Operator:

```
var JSONObj = { "bookname ":"VB BLACK BOOK", "price":500 };
```

This is an example which shows creation of an object in javascript using JSON, save the below code as **json_object.htm**:

```
<html>
<head>
<title>Creating Object JSON with JavaScript</title>
<script language="javascript" >

  var JSONObj = { "name" : "tutorialspoint.com", "year"  : 2005 };
  document.write("<h1>JSON with JavaScript example</h1>");
  document.write("<br>");
  document.write("<h3>Website Name="+JSONObj.name+"</h3>");
  document.write("<h3>Year="+JSONObj.year+"</h3>");

</script>
</head>
<body>
</body>
</html>
```

- **Creating Array Objects**

Below example shows creation of an array object in javascript using JSON, save the below code as **json_array_object.htm**:

```
<html>
<head>
<title>Creation of array object in javascript using JSON</title>
<script language="javascript" >

document.writeln("<h2>JSON array object</h2>");

var books = { "Pascal" : [
     { "Name"  : "Pascal Made Simple", "price" : 700 },
     { "Name"  : "Guide to Pascal", "price" : 400 }
   ],
```

```
  "Scala"  : [
    { "Name"  : "Scala for the Impatient", "price" : 1000 },
    { "Name"  : "Scala in Depth", "price" : 1300 }
  ]
}

var i = 0
document.writeln("<table border='2'><tr>");
for(i=0;i<books.Pascal.length;i++)
{
  document.writeln("<td>");
  document.writeln("<table border='1' width=100 >");
  document.writeln("<tr><td><b>Name</b></td><td width=50>"
  + books.Pascal[i].Name+"</td></tr>");
  document.writeln("<tr><td><b>Price</b></td><td width=50>"
  + books.Pascal[i].price +"</td></tr>");
  document.writeln("</table>");
  document.writeln("</td>");
}

for(i=0;i<books.Scala.length;i++)
{
  document.writeln("<td>");
  document.writeln("<table border='1' width=100 >");
  document.writeln("<tr><td><b>Name</b></td><td width=50>"
  + books.Scala[i].Name+"</td></tr>");
  document.writeln("<tr><td><b>Price</b></td><td width=50>"
  + books.Scala[i].price+"</td></tr>");
  document.writeln("</table>");
  document.writeln("</td>");
}
document.writeln("</tr></table>");
</script>
</head>
<body>
</body>
</html>
```

FAQs:

1. What is difference between XML amd JSON?
2. Write a program to demonstrate simple JSON objects using java script
3. Write a program for creation of array object in java script using JSON

Oral/Review Questions:

1. Explain what is JSON?
2. What is the file extension name of JSON?
3. How many languages, including in JSON?
4. What is the rule for JSON syntax rules? Explain with an example of JSON object?

ASSIGNMENT NUMBER: B7

Revised On: 15/06/2018

TITLE	Study and demonstrate the use of encoding and decoding JSON objects using Java/Perl/PHP/Python/Ruby.
PROBLEM STATEMENT /DEFINITION	Study and demonstrate the use of encoding and decoding JSON objects using Java/Perl/PHP/Python/Ruby.
OBJECTIVE	• To understand and implement encoding and decoding of JSON objects.
S/W PACKAGES AND HARDWARE APPARATUS USED	• Eclipse • JAVA/Python • PC with the configuration as Latest Version of 64 bit Operating Systems, Open Source Fedora-GHz. 8 G.B. RAM, 500 G.B. HDD, 15"Color Monitor, Keyboard, Mouse
REFERENCES	.**http://www.tutorialspoint.com/json/**
STEPS	Refer to details
INSTRUCTIONS FOR WRITING JOURNAL	• **Title** • **Problem Definition** • **Learning Objectives** • **Theory** • **Class Diagram/ER Diagram** • **Test cases**

	• **Program Listing** • **Output** • **Conclusion**

Aim: Encode and Decode JSON Objects using Java/Perl/PHP/Python/Ruby.

Problem Statment :Study and demonstrate the use of encoding and decoding JSON objects using Java/Perl/PHP/Python/Ruby.

Requirements:

- Computer System with Windows/Linux/Open Source Operating System.
- JavaScript
- JAVA
- PHP
- PYTHON

Theory:

JSON extension is bundled with PHP by default from version 5.2.0 so there is no need of any special environment.

JSON Functions:

1.json_encode: It returns the JSON representation of a value.

2.json_decode: It decodes a JSON string.

3.json_last_error: It returns the last error occurred.

Encoding:

json_encode() function is used for encoding which returns JSON representation of a value.

Syntax:

string json_encode ($value [, $options = 0])

The value parameter specifies value being specified. It works only with UTF-8 encoded data.

The options parameter specifies the a bitmask consisting of

JSON_HEX_QUOT,JSON_HEX_TAG,JSON_HEX_AMP,JSON_HEX_APOS,JSON_NUM

ERIC_CHECK,JSON_PRETTY_PRINT,JSON_UNESCAPED_SLASHES,
JSON_FORCE_OBJECT.

Example:

The following PHP code

```
<?php

class Emp {

public $name = "";

public

$hobbies = "";

public $birthdate = "";

}

$e = new Emp();

$e->name = "sachin";

$e->hobbies = "sports";
```

$e->birthdate = date('m/d/Y h:i:s a', strtotime("8/5/1974 11:20:03"));

echo json_encode($e);?>

can be encoded to JSON object

{"name":"sachin","hobbies

":"sports","birthdate":"08/05/1974

11:20:03 pm"}

Decoding:

json_decode () function is used for decoding JSON object in to PHP.

Syntax:

json_decode ($json [,$assoc = false [, $depth = 511 [, $options = 0]]])

Parameters:

6. json_string :It is encoded string which must be UTF-8 encoded data.
7. assoc :It is a boolean type parameter, when set to TRUE, returned objects will be converted into associative arrays.
8. depth:It is an integer type parameter which specifies recursion depth
9. options: It is an integer type bitmask of JSON decode. It supports JSON_BIGINT_AS_STRING

Example:

The following JSON object

<?php

$json = '{"a":1,"b":2,"c":3,"d":4,"e":5}';

```
var_dump(json_decode($json));

var_dump(json_decode($json, true));

?>
```

```
Can be decoded into object(stdClass)#1 (5) {

["a"] => int(1)

["b"] => int(2)

["c"] => int(3)

["d"] => int(4)

["e"] => int(5)

}

array(5) {

["a"] => int(1)

["b"] => int(2)

["c"] => int(3)

["d"] => int(4)

["e"] => int(5)

}
```

FAQs:

1. Implement encoding/decoding of JSON objects using JAVA.

2. Define encoding and decoding of objects in general.

3. Whether encoding/decoding supports simple JSON objects or JSON Array objects or both. Explain with suitable example.

Oral/Review Questions:

1. What is JSON?
2. How create JSON object in Java?
3. What is mean by encoding/decoding?

ASSIGNMENT NUMBER: C1

Revised On: 15/06/2018

TITLE	Implement MONGO DB database connectivity with PHP/ python/Java. Implement Database navigation operations (add, delete, edit,) using ODBC/JDBC
PROBLEM STATEMENT /DEFINITION	To study and implement database connectivity and perform operations on it.
OBJECTIVE	• Understand the concept of database. • Understand the concept of mongo db. • Understand the database operations.
S/W PACKAGES AND HARDWARE APPARATUS USED	• Mongo db • Java / Python / PHP • PC with the configuration as Latest Version of 64 bit Operating Systems,Open Source Fedora-GHz. 8 G.B. RAM, 500 G.B. HDD, 15"Color Monitor, Keyboard, Mouse
REFERENCES	• 1.Silberschatz A., Korth H., Sudarshan S., "Database System Concepts", 5th Edition, McGraw Hill Publishers, 2002, ISBN 0-07-120413-X • Connally T., Begg C., "Database Systems", 3rd Edition, Pearson Education, 2002, ISBN 81-7808-861-4 • mysql.com/docs/mysql-tutorial-excerpt-5.1-en.pdf
STEPS	Refer to details

INSTRUCTIONS FOR **WRITING JOURNAL**	• **Date** • **Title** • **Problem Definition** • **Learning Objectives** • **Learning Outcomes** • **Theory** • **Class Diagram/ER Diagram** • **Test cases** • **Program Listing** • **Output** • **Conclusion**

Aim: Implement Mongo DB database connectivity with PHP/ python/Java. Implement Database navigation operations (add, delete, edit,) using ODBC/JDBC.

Pre-requisite:

Basic knowledge of database, java/python programming and php.

Learning Objectives:

- To understand & implement the database connectivity and perform operations.

Learning Outcomes:

The students will be able to

- Implement the database using Mongo DB.
- Implement and perform different queries on database.

Theory:

MongoDB (from *hu**mongo**us*) is a free and open-source cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with schemas. MongoDB is developed by MongoDB Inc., and is published under a combination of the GNU Affero General Public License and the Apache License.

Connect to Database java:

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

Following is the syntax to connect to the database −

```
    // Creating a Mongo client
    MongoClient mongo = new MongoClient( "localhost" , 27017 );

    // Creating Credentials
    MongoCredential credential;
    credential = MongoCredential.createCredential("sampleUser", "myDb",
      "password".toCharArray());
    System.out.println("Connected to the database successfully");

    // Accessing the database
    MongoDatabase database = mongo.getDatabase("myDb");
    System.out.println("Credentials ::"+ credential);
  }
}
```

Database connection with python:

```
>>> import pymongo
```

$ mongod

Making a Connection with MongoClient

The first step when working with **PyMongo** is to create a MongoClient to the running **mongod** instance. Doing so is easy:

```
>>> from pymongo import MongoClient
>>> client = MongoClient()
```

The above code will connect on the default host and port. We can also specify the host and port explicitly, as follows:

```
>>> client = MongoClient('localhost', 27017)
```

Or use the MongoDB URI format:

>>> client = MongoClient('mongodb://localhost:27017/')

FAQs:

- Explain the RDBMS?
- Explain commands to insert,delete,modify into database table ?

Oral/Review Questions:

- What is the difference between DBMS and RDBMS ?
- Which of the following will be used to modify entries in database (insert, delete, update) command.

ASSIGNMENT NUMBER: C2

Revised On: 15/06/2018

TITLE	Implement MYSQL/Oracle database connectivity with PHP/ python/Java. Implement Database navigation operations (add, delete, edit,) using ODBC/JDBC
PROBLEM STATEMENT /DEFINITION	To study and implement database connectivity and perform operations on it.
OBJECTIVE	• Understand the concept of database. • Understand the concept of ODBC/JDBC • Understand the database operations
S/W PACKAGES AND HARDWARE APPARATUS USED	• MY SQL • Java / Python / PHP • PC with the configuration as Latest Version of 64 bit Operating Systems,Open Source Fedora-GHz. 8 G.B. RAM, 500 G.B. HDD, 15"Color Monitor, Keyboard, Mouse
REFERENCES	• 1.Silberschatz A., Korth H., Sudarshan S., "Database System Concepts", 5th Edition, McGraw Hill Publishers, 2002, ISBN 0-07-120413-X • Connally T., Begg C., "Database Systems", 3rd Edition, Pearson Education, 2002, ISBN 81-7808-861-4 • mysql.com/docs/mysql-tutorial-excerpt-5.1-en.pdf
STEPS	Refer to details

INSTRUCTIONS FOR **WRITING JOURNAL**	• **Date** • **Title** • **Problem Definition** • **Learning Objectives** • **Learning Outcomes** • **Theory** • **Class Diagram/ER Diagram** • **Test cases** • **Program Listing** • **Output** • **Conclusion**

Aim: Implement MYSQL/Oracle database connectivity with PHP/ python/Java. Implement Database navigation operations (add, delete, edit,) using ODBC/JDBC.

Pre-requisite:

Basic knowledge of database, java/python programming and php.

Learning Objectives:

- To understand & implement the database connectivity and perform operations.

Learning Outcomes:

The students will be able to

- Implement the database using MySQL.

- Implement and perform different queries on database.

Theory:

MySQL is the most popular Open Source Relational SQL Database Management System. MySQL is one of the best RDBMS being used for developing various web-based software applications. MySQL is developed, marketed and supported by MySQL AB, which is a Swedish company. This tutorial will give you a quick start to MySQL and make you comfortable with MySQL programming.

Connectivity to Database

```
$dbhost = 'localhost:3306';
$dbuser = 'guest';
$dbpass = 'guest123';
$conn = mysql_connect($dbhost, $dbuser, $dbpass);
if(! $conn ) {
die('Could not connect: ' . mysql_error());
}
echo 'Connected successfully';
```

Insert into Database

```
$sql = "INSERT INTO tutorials_tbl ".
   "(tutorial_title,tutorial_author, submission_date) "."VALUES ".
   "('$tutorial_title','$tutorial_author','$submission_date')";
```

Delete from Database

```
$sql = 'DELETE FROM tutorials_tbl' WHERE tutorial_id=3';
```

Update into Database

```
$sql = 'UPDATE tutorials_tbl SET tutorial_title="Learning JAVA"  WHERE
tutorial_id=3';
```

FAQs:

- Explain the RDBMS?
- Explain commands to insert,delete,modify into database table ?

Oral/Review Questions:

- What is the difference between DBMS and RDBMS ?
- Which of the following will be used to modify entries in database (insert, delete, update) command.

ASSIGNMENT NUMBER: C3

Revised On: 15/06/2018

TITLE	Using the database concepts covered in Part-I & Part-II & connectivity concepts covered in Part C, students in group are expected to design and develop database application.
PROBLEM STATEMENT /DEFINITION	To design and develop database application with following details: • Design Entity Relationship Model, Relational Model, Database Normalization
OBJECTIVE	• Understand the concept of database. • Develop a database application
S/W PACKAGES AND HARDWARE APPARATUS USED	• MY SQL • Java / PythFront End : Java/Perl/PHP/Python/Ruby/.net Backend : MongoDB/MYSQL/Oracle Database Connectivity : ODBC/JDBC • PC with the configuration as Latest Version of 64 bit Operating Systems,Open Source Fedora-GHz. 8 G.B. RAM, 500 G.B. HDD, 15"Color Monitor, Keyboard, Mouseon / PHP
REFERENCES	• Silberschatz A., Korth H., Sudarshan S., "Database System Concepts", 5th Edition, McGraw Hill Publishers, 2002, ISBN 0-07-120413-X • Connally T., Begg C., "Database Systems", 3rd Edition, Pearson Education, 2002, ISBN 81-7808-861-4 • mysql.com/docs/mysql-tutorial-excerpt-5.1-en.pdf

STEPS	Refer to details
INSTRUCTIONS FOR **WRITING JOURNAL**	• **Date** • **Title** • **Problem Definition** • **Learning Objectives** • **Learning Outcomes** • **Theory** • **Class Diagram/ER Diagram** • **Test cases** • **Program Listing** • **Output** • **Conclusion**

Aim: Using the database concepts covered in Part-I & Part-II & connectivity concepts covered in Part C, students in group are expected to design and develop database application.

Pre-requisite:

Basic knowledge of database, java/python programming and php.

Learning Objectives:

- To understand, design & implement the database application.

Learning Outcomes:

The students will be able to

- Analyze and design database applications
- Implement the database using Open source relational Database management system.
- Develop database application

Guidelines for Miniproject:

- 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 Modelling features, Data Dictionary, Relational Database Design, Database Normalization, Graphical User Interface, Source Code, Testing document, Conclusion.
- Students should follow the following Requirement Gathering and Scope finalization

1. Database Analysis and Design:

2. Design Entity Relationship Model, Relational Model, Database Normalization

3. Implementation :

   - Front End : Java/Perl/PHP/Python/Ruby/.net

   - Backend : MongoDB/MYSQL/Oracle

- Database Connectivity : ODBC/JDBC

Testing: Data Validation

FAQs:

- Explain the RDBMS?
- Explain commands to insert, delete, modify into database table?

Oral/Review Questions:

- What is the difference between DBMS and RDBMS?
- What are SQL, PLSQL, and NOSQL?
- How to do JDBC and MySql connectivity?
- What is 2-tire and 3tire architecture?
- Which of the following will be used to modify entries in database (insert, delete, update) command.