Module 3: Database & SQL

Module Overview

In this module, students will be able to familiarize with database connectivity.

Module Objective

At the end of this module, students should be able to demonstrate appropriate knowledge, and show an understanding of the following:

- Dramatize the structure of JDBC.
- Illustrate the queries occurred.
- Demonstrate Connection management.

Design of JDBC

JDBC stands for **Java Database Connectivity**, which is a standard Java API for database-independent connectivity between the Java programming language and a wide range of databases.

The JDBC library includes APIs for each of the tasks mentioned below that are commonly associated with database usage:

- Making a connection to a database.
- Creating SQL or MySQL statements.
- Executing SQL or MySQL queries in the database.
- Viewing & Modifying the resulting records.

Fundamentally, JDBC is a specification that provides a complete set of interfaces that allows for portable access to an underlying database. Java can be used to write different types of executables, such as −

- Java Applications
- Java Applets
- Java Servlets
- Java Server Pages (JSPs)

- Enterprise JavaBeans (EJBs).

All of these different executables are able to use a JDBC driver to access a database and take advantage of the stored data.

JDBC provides the same capabilities as ODBC, allowing Java programs to contain database-independent code.

JDBC Architecture

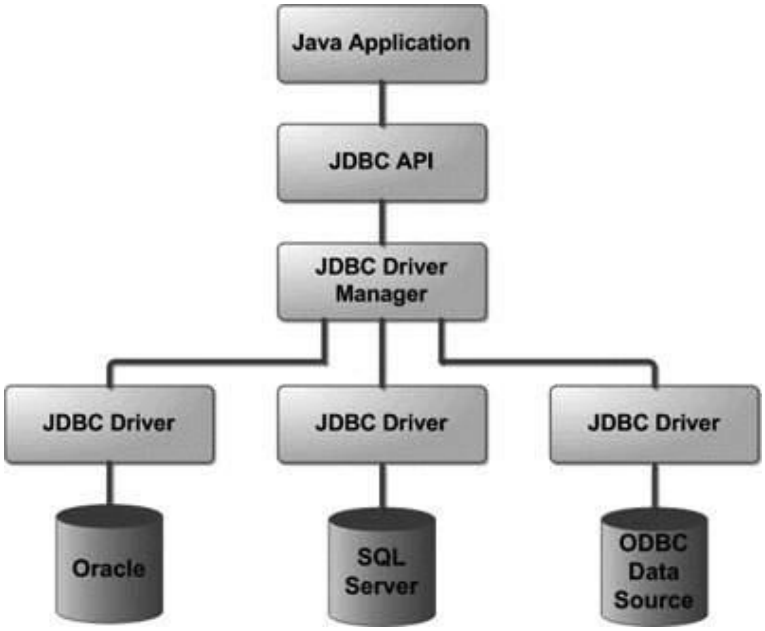
The JDBC API supports both two-tier and three-tier processing models for database access but in general, JDBC Architecture consists of two layers −

- JDBC API: This provides the application-to-JDBC Manager connection.
- JDBC Driver API: This supports the JDBC Manager-to-Driver Connection.

The JDBC API uses a driver manager and database-specific drivers to provide transparent connectivity to heterogeneous databases.

The JDBC driver manager ensures that the correct driver is used to access each data source. The driver manager is capable of supporting multiple concurrent drivers connected to multiple heterogeneous databases.

Following is the architectural diagram, which shows the location of the driver manager with respect to the JDBC drivers and the Java application –



Now we shall see the interfaces and classes provided by JDBC API, as follows:

- **DriverManager**: This class manages a list of database drivers. Matches connection requests from the java application with the proper database driver using communication sub protocol. The first driver that recognizes a certain subprotocol under JDBC will be used to establish a database Connection.

- **Driver**: This interface handles the communications with the database server. You will interact directly with Driver objects very rarely. Instead, you use DriverManager objects, which manages objects of this type. It also abstracts the details associated with working with Driver objects.

- **Connection**: This interface with all methods for contacting a database. The connection object represents communication context, i.e., all communication with database is through connection object only.

- **Statement**: You use objects created from this interface to submit the SQL statements to the database. Some derived interfaces accept parameters in addition to executing stored procedures.

- **ResultSet**: These objects hold data retrieved from a database after you execute an SQL query using Statement objects. It acts as an iterator to allow you to move through its data.

- **SQLException**: This class handles any errors that occur in a database application.

The JDBC Packages

The java.sql and javax.sql are the primary packages for JDBC 4.0. This is the latest JDBC version at the time of writing this tutorial. It offers the main classes for interacting with your data sources.

The new features in these packages include changes in the following areas −

- Automatic database driver loading.
- Exception handling improvements.
- Enhanced BLOB/CLOB functionality.
- Connection and statement interface enhancements.
- National character set support.
- SQL ROWID access.
- SQL 2003 XML data type support.
- Annotations.

JDBC Installation

To start developing with JDBC, you should setup your JDBC environment by following the steps shown below.

Install JDK first from Official Java website. Then follow the below points it required:

- **JAVA_HOME**: This environment variable should point to the directory where you installed the JDK, e.g. C:\Program Files\Java\jdk1.x.0.
- **CLASSPATH**: This environment variable should have appropriate paths set, e.g. C:\Program Files\Java\jdk1.x.0_20\jre\lib.
- **PATH**: This environment variable should point to appropriate JRE bin, e.g. C:\Program Files\Java\jre1.x.0_20\bin.

The most important thing you will need, of course is an actual running database with a table that you can query and modify.

MySQL is an open source database. You can download it from MySQL Official Site. We recommend downloading the full Windows installation.

In addition, download and install MySQL Administrator as well as MySQL Query Browser. These are GUI based tools that will make your development much easier.

Finally, download and unzip MySQL Connector/J (the MySQL JDBC driver) in a convenient directory. For the purpose of this tutorial we will assume that you have installed the driver at C:\Program Files\MySQL\mysql-connector-java-x.y.

Accordingly, set CLASSPATH variable to C:\Program Files\MySQL\mysql-connector-java-x.y.\mysql-connector-java-x.y.-bin.jar. Your driver version may vary based on your installation.

The Structured Query Language

What is SQL?

SQL stands for **Structured Query Language**. SQL tutorial provides basic and advanced concepts of SQL. Our SQL tutorial is designed for beginners and professionals.

SQL is used to perform operations on the records stored in the database such as updating records, deleting records, creating and modifying tables, views, etc.

SQL is just a query language; it is not a database. To perform SQL queries, you need to install any database, for example, Oracle, MySQL, MongoDB, PostGre SQL, SQL Server, DB2, etc.

SQL is required:

- To create new databases, tables and views
- To insert records in a database
- To update records in a database
- To delete records from a database

- To retrieve data from a database

What is Database?

A database is an organized collection of data, so that it can be easily accessed and managed.

You can organize data into tables, rows, columns, and index it to make it easier to find relevant information. Database handlers create a database in such a way that only one set of software program provides access of data to all the users.

The main purpose of the database is to operate a large amount of information by storing, retrieving, and managing data. There are many dynamic websites on the World Wide Web nowadays which are handled through databases. For example, a model that checks the availability of rooms in a hotel. It is an example of a dynamic website that uses a database. There are many databases available like MySQL, Sybase, Oracle, MongoDB, Informix, PostgreSQL, SQL Server, etc. Modern databases are managed by the database management system (DBMS).

SQL or Structured Query Language is used to operate on the data stored in a database. SQL depends on relational algebra and tuple relational calculus.

A cylindrical structure is used to display the image of a database.

SQL Syntax

SQL follows some unique set of rules and guidelines called syntax. Here, we are providing all the basic SQL syntax:

- SQL is not case sensitive. Generally, SQL keywords are written in **uppercase**.
- SQL statements are dependent on text lines. We can place a single SQL statement on one or multiple text lines.
- You can perform most of the action in a database with SQL statements.
- SQL depends on relational algebra and tuple relational calculus.

SQL statements are started with any of the SQL commands/keywords like SELECT, INSERT, UPDATE, DELETE, ALTER, DROP etc. and the statement ends with a semicolon (;).

Example of SQL statement:

```
SELECT "column_name" FROM "table_name";
```

Semicolon is used to **separate** SQL statements. It is a standard way to separate SQL statements in a database system in which more than one SQL statements are used in the same call.

Following are some of the important SQL commands:

SELECT: it extracts data from a database.

UPDATE: it updates data in database.

DELETE: it deletes data from database.

CREATE TABLE: it creates a new table.

ALTER TABLE: it is used to modify the table.

DROP TABLE: it deletes a table.

CREATE DATABASE: it creates a new database.

ALTER DATABASE: It is used to modify a database.

INSERT INTO: it inserts new data into a database.

CREATE INDEX: it is used to create an index (search key).

DROP INDEX: it deletes an index.

Database Tables

A database most often contains one or more tables. Each table is identified by a name (e.g. "Customers" or "Orders"). Tables contain records (rows) with data.

Tips : SQL keywords **are NOT case sensitive**: select is the same as SELECT

Semicolon after SQL Statements?

Some database systems require a semicolon at the end of each SQL statement.

Semicolon is the standard way to separate each SQL statement in database systems that allow more than one SQL statement to be executed in the same call to the server.

Data Types

Data types define the nature of the data that can be stored in a particular column of a table

MySQL has **3** main categories of data types namely

1. Numeric,
2. Text
3. Date/time.

1.Numeric Data types

Numeric data types are used to store numeric values. It is very important to make sure range of your data is between lower and upper boundaries of numeric data types.

TINYINT( )	-128 to 127 normal 0 to 255 UNSIGNED.
SMALLINT( )	-32768 to 32767 normal 0 to 65535 UNSIGNED.
MEDIUMINT( )	-8388608 to 8388607 normal 0 to 16777215 UNSIGNED.
INT( )	-2147483648 to 2147483647 normal 0 to 4294967295 UNSIGNED.
BIGINT( )	-9223372036854775808 to 9223372036854775807 normal 0 to 18446744073709551615 UNSIGNED.
FLOAT	A small approximate number with a floating decimal point.
DOUBLE(,)	A large number with a floating decimal point.

DECIMAL(,)	A DOUBLE stored as a string , allowing for a fixed decimal point. Choice for storing currency values.

2.Text Data Types

As data type category name implies these are used to store text values. Always make sure you length of your textual data do not exceed maximum lengths.

CHAR( )	A fixed section from 0 to 255 characters long.
VARCHAR( )	A variable section from 0 to 255 characters long.
TINYTEXT	A string with a maximum length of 255 characters.
TEXT	A string with a maximum length of 65535 characters.
BLOB	A string with a maximum length of 65535 characters.
MEDIUMTEXT	A string with a maximum length of 16777215 characters.
MEDIUMBLOB	A string with a maximum length of 16777215 characters.
LONGTEXT	A string with a maximum length of 4294967295 characters.
LONGBLOB	A string with a maximum length of 4294967295 characters.

3.Date / Time

DATE	YYYY-MM-DD
DATETIME	YYYY-MM-DD HH:MM:SS
TIMESTAMP	YYYYMMDDHHMMSS
TIME	HH:MM:SS

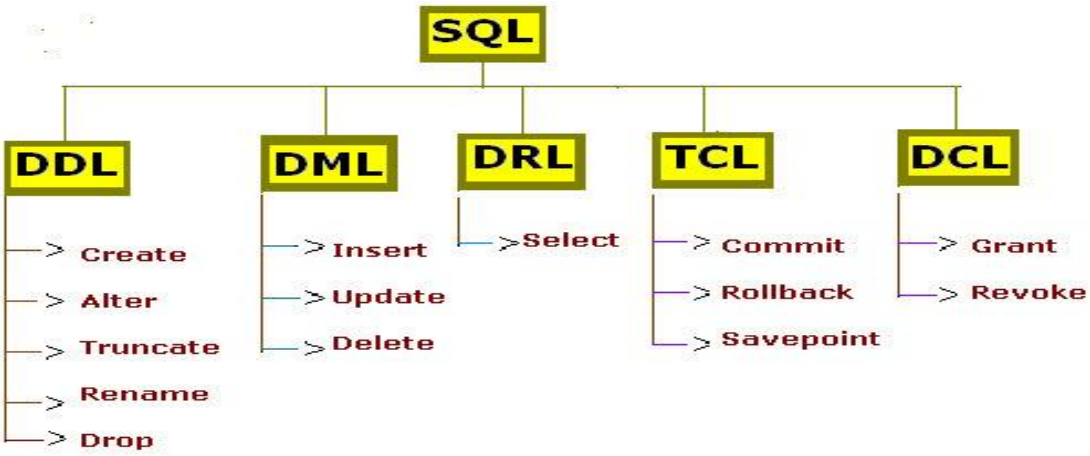
Apart from above there are some other data types in MySQL.

ENUM	To store text value chosen from a list of predefined text values
SET	This is also used for storing text values chosen from a list of predefined text values. It can have multiple values.
BOOL	Synonym for TINYINT(1), used to store Boolean values
BINARY	Similar to CHAR, difference is texts are stored in binary format.
VARBINARY	Similar to VARCHAR, difference is texts are stored in binary format.

SQL Commands

SQL commands are mainly categorized into five categories as discussed below:

- DDL
- DML
- DRL
- DCL
- TCL



DDL(Data Definition Language) :

DDL or Data Definition Language actually consists of the SQL commands that can be used to define the database schema. It simply deals with descriptions of the database schema and is used to create and modify the structure of database objects in database.

Examples of DDL commands:

1)CREATE – is used to create the database or its objects (like table, index, function, views, store procedure and triggers). There are two CREATE statements available in SQL:

1. CREATE DATABASE
2. CREATE TABLE

i.CREATE DATABASE

A Database is defined as a structured set of data. So, in SQL the very first step to store the data in a well structured manner is to create a database. The CREATE DATABASEstatement is used to create a new database in SQL.

Syntax:

```
CREATE DATABASE database_name;
```

database_name: name of the database.

Example:

This query will create a new database in SQL and name the database as my_database.

```
CREATE DATABASE my_database;
```

ii.CREATE TABLE

The CREATE TABLE statement is used to create a table in SQL. We know that a table comprises of rows and columns. So while creating tables we have to provide all the information to SQL about the names of the columns, type of data to be stored in columns, size of the data etc. Let us now dive into details on how to use CREATE TABLE statement to create tables in SQL.

Syntax:

```
CREATE TABLE table_name
(
column1 data_type(size),
column2 data_type(size),
column3 data_type(size),
....
);
```

table_name: name of the table.
column1 name of the first column.
data_type: Type of data we want to store in the particular column.
For example,int for integer data.
size: Size of the data we can store in a particular column.
For example if for a column we specify the data_type as int and size as 10 then this column can store an integer number of maximum 10 digits.

ExampleQuery:
This query will create a table named Students with three columns, ROLL_NO, NAME and SUBJECT.

```
CREATE TABLE Students
(
ROLL_NO int(3),
NAME varchar(20),
SUBJECT varchar(20),
);
```

2)<u>DROP</u> – is used to delete objects from the database.
Examples:

```
DROP TABLE table_name;
```

table_name: Name of the table to be deleted.

DROP DATABASE database_name;
database_name: Name of the database to be deleted.

If you want to delete an existing database <test>, then the DROP DATABASE statement would be as shown below −

```
SQL> DROP DATABASE testDB;
```

NOTE − Be careful before using this operation because by deleting an existing database would result in loss of complete information stored in the database.

Similarly, use delete a table in database using DROP.

3)ALTER-is used to alter the structure of the database.

ALTER TABLE is used to add, delete/drop or modify columns in the existing table. It is also used to add and drop various constraints on the existing table.

i.ALTER TABLE – ADD

ADD is used to add columns into the existing table.

Syntax:

```
ALTER TABLE table_name
        ADD (Columnname_1  datatype,
        Columnname_2  datatype,
        …
        Columnname_n  datatype);
```

ii.ALTER TABLE – DROP

DROP COLUMN is used to drop column in a table. Deleting the unwanted columns from the table.

Syntax:

```
ALTER TABLE table_name
DROP COLUMN column_name;
```

iii.ALTER TABLE-MODIFY

It is used to modify the existing columns in a table. Multiple columns can also be modified at once.

Syntax:

```
ALTER TABLE table_name
MODIFY column_name column_type;
```

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|
+----+----------+-----+-----------+----------+
```

Following is the example to ADD a New Column to an existing table −

```
ALTER TABLE CUSTOMERS ADD SEX char(1);
```

Now, the CUSTOMERS table is changed and following would be output from the SELECT statement.

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

Following is the example to DROP sex column from the existing table.

```
ALTER TABLE CUSTOMERS DROP SEX;
```

Now, the CUSTOMERS table is changed and following would be the output from the SELECT statement.

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

4)TRUNCATE–is used to remove all records from a table, including all spaces allocated for the records are removed.

Syntax:

```
TRUNCATE TABLE  table_name;

table_name: Name of the table to be truncated.
```

RENAME Command –It is used to rename an object existing in the database.

Syntax:

```
ALTER TABLE table_name
RENAME TO new_table_name;
```

DML(Data Manipulation Language) :

The SQL commands that deals with the manipulation of data present in database belong to DML or Data Manipulation Language and this includes most of the SQL statements.

Examples of DML:

1)SELECT – is used to retrieve data from the a database.

Basic Syntax:

```
SELECT column1,column2 FROM table_name
column1 , column2: names of the fields of the table

table_name: from where we want to fetch
```

This query will return all the rows in the table with fields column1 , column2.

- To fetch the entire table or all the fields in the table:

```
SELECT * FROM table_name;
```

- Query to fetch the fields ROLL_NO, NAME, AGE from the table Student:

```
SELECT ROLL_NO, NAME, AGE FROM Student;
```

2)INSERT – is used to insert data into a table.
There are two ways of using INSERT INTO statement for inserting rows:

Only values: First method is to specify only the value of data to be inserted without the column names.
Syntax:

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

table_name: name of the table.
value1, value2, .. : value of first column, second column,... for the new record
```

Column names and values both: In the second method we will specify both the columns which we want to fill, and their corresponding values as shown below:
Syntax:

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

table_name: name of the table.
column1: name of first column, second column ...
```

value1, value2, value3 : value of first column, second column,... for the new record

3)UPDATE – is used to update existing data within a table.
Basic Syntax

```
UPDATE table_name SET column1 = value1, column2 = value2,...
WHERE condition;

table_name: name of the table
column1: name of first , second, third column....
value1: new value for first, second, third column....
condition: condition to select the rows for which the
values of columns need to be updated.
```

4)DELETE – is used to delete records from a database table.
Basic Syntax:

DELETE FROM table_name WHERE some_condition;

table_name: name of the table

some_condition: condition to choose particular record.

Difference between Delete, Drop and Truncate:

DELETE Statement:
The DELETE command is used to remove rows from a table. A WHERE clause can be used to only remove some rows. If no WHERE condition is specified, all rows will be removed. After performing a DELETE operation you need to COMMIT or ROLLBACK the transaction to make the change permanent or to undo it. Note that this operation will cause all DELETE triggers on the table to fire.

TRUNCATE statement:
TRUNCATE removes **all rows** from a table. The operation cannot be rolled back and no triggers will be fired.

DROP statement:
The DROP command removes a table from the database. All the tables' rows, indexes and privileges will also be removed. No DML triggers will be fired. The operation cannot be rolled back.

DRL/DSL(Data Retrieval Language/ Data Selection Language) :

- DRL/DSL stands for Data Retrieval Language/Data Selection Language.
- It is a set commands which are used to retrieve data from database server.
- It manipulates the data in database for display purpose like aggregate function.
- In DRL/DSL, for accessing the data it uses the DML command that is SELECT.
- The SELECT command allows database users to retrieve the specific information they desire from an operational database.

SELECT clause has many optional clauses are as follow;

Clause	Description
FROM	It is used for selecting a table name in a database.
WHERE	It specifies which rows to retrieve.

GROUP BY	It is used to arrange the data into groups.
HAVING	It selects among the groups defined by the GROUP BY clause.
ORDER BY	It specifies an order in which to return the rows.
AS	It provides an alias which can be used to temporarily rename tables or columns.

Syntax:
SELECT {* | column_name1, column_name2, . . . , column_name_n}
FROM <list_of_tablename>
WHERE <condition>;

Example

```
SELECT * FROM employee
WHERE salary >=10000;

OR

SELECT eid, ename, age, salary
WHERE salary >=10000;
```

DCL(Data Control Language) :

DCL includes commands such as GRANT and REVOKE which mainly deals with the rights, permissions and other controls of the database system.
Examples of DCL commands:
1)GRANT-gives user's access privileges to database.
2)REVOKE-withdraw user's access privileges given by using the GRANT command.

- **Allow a User to create session**

When we create a user in SQL, it is not even allowed to login and create a session until and unless proper permissions/priviliges are granted to the user.

Following command can be used to grant the session creating priviliges.

```
GRANT CREATE SESSION TO username;
```

- **Allow a User to create table**

To allow a user to create tables in the database, we can use the below command,

```
GRANT CREATE TABLE TO username;
```

- **Provide user with space on tablespace to store table**

Allowing a user to create table is not enough to start storing data in that table. We also must provide the user with priviliges to use the available tablespace for their table and data.

```
ALTER USER username QUOTA UNLIMITED ON SYSTEM;
```

The above command will alter the user details and will provide it access to unlimited tablespace on system.

NOTE: Generally unlimited quota is provided to Admin users.

- Grant all privilege to a User

sysdba is a set of priviliges which has all the permissions in it. So if we want to provide all the privileges to any user, we can simply grant them the sysdba permission.

```
GRANT sysdba TO username
```

- **Grant permission to create any table**

Sometimes user is restricted from creating come tables with names which are reserved for system tables. But we can grant privileges to a user to create any table using the below command,

```
GRANT CREATE ANY TABLE TO username
```

- **Grant permission to drop any table**

As the title suggests, if you want to allow user to drop any table from the database, then grant this privilege to the user,

```
GRANT DROP ANY TABLE TO username
```

- **To take back Permissions**

And, if you want to take back the privileges from any user, use the REVOKE command.

```
REVOKE CREATE TABLE FROM username
```

TCL(Transaction Control Language) :

TCL commands deals with the transaction within the database.
Examples of TCL commands:
1)COMMIT– commits a Transaction.
When we use any DML command like INSERT, UPDATE or DELETE, the changes made by these commands are not permanent, until the current session is closed, the changes made by these commands can be rolled back.

To avoid that, we use the COMMIT command to mark the changes as permanent.

Following is commit command's syntax,

```
COMMIT;
```

2)ROLLBACK– rollbacks a transaction in case of any error occurs.
This command restores the database to last commited state. It is also used with SAVEPOINT command to jump to a savepoint in an ongoing transaction.

If we have used the UPDATE command to make some changes into the database, and realise that those changes were not required, then we can use the ROLLBACK command to rollback those changes, if they were not commited using the COMMIT command.

Following is rollback command's syntax,

```
ROLLBACK TO savepoint_name;
```

3)SAVEPOINT–sets a savepoint within a transaction.
SAVEPOINT command is used to temporarily save a transaction so that you can rollback to that point whenever required.

Following is savepoint command's syntax,

```
SAVEPOINT savepoint_name;
```

In short, using this command we can name the different states of our data in any table and then rollback to that state using the ROLLBACK command whenever required.

Using Savepoint and Rollback

Following is the table class,

Id	name
1	Abhi
2	Adam
4	Alex

Lets use some SQL queries on the above table and see the results.

```
INSERT INTO class VALUES(5, 'Rahul');
COMMIT;
UPDATE class SET name = 'Abhijit' WHERE id = '5';
SAVEPOINT A;
INSERT INTO class VALUES(6, 'Chris');
SAVEPOINT B;
INSERT INTO class VALUES(7, 'Bravo');
SAVEPOINT C;
SELECT * FROM class;
```

NOTE: SELECT statement is used to show the data stored in the table.

The resultant table will look like,

Id	name

1	Abhi
2	Adam
4	Alex
5	Abhijit
6	Chris
7	Bravo

Now let's use the ROLLBACK command to roll back the state of data to the savepoint B.

```
ROLLBACK TO B;
SELECT * FROM class;
```

Now our class table will look like,

Id	name
1	Abhi
2	Adam
4	Alex
5	Abhijit

6	Chris

Now let's again use the ROLLBACK command to roll back the state of data to the savepoint A

```
ROLLBACK TO A;
SELECT * FROM class;
```

Now the table will look like,

Id	name
1	Abhi
2	Adam
4	Alex
5	Abhijit

So now you know how the commands COMMIT, ROLLBACK and SAVEPOINT works.

Expressions

An expression is a combination of one or more values, operators and SQL functions that evaluate to a value. These SQL EXPRESSIONs are like formulae and they are written in query language. You can also use them to query the database for a specific set of data.

Consider the basic syntax of the SELECT statement as follows −

```
SELECT column1, column2, columnN
FROM table_name
WHERE [CONDITION|EXPRESSION];
```

There are different types of SQL expressions, which are mentioned below −

- Boolean
- Numeric
- Date

1. Boolean Expressions

SQL Boolean Expressions fetch the data based on matching a single value. Following is the syntax −

```
SELECT column1, column2, columnN
FROM table_name
WHERE SINGLE VALUE MATCHING EXPRESSION;
```

Consider the CUSTOMERS table having the following records −

```
SQL> SELECT * FROM CUSTOMERS;

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

| 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|

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

7 rows inset(0.00 sec)
```

The following table is a simple example showing the usage of various SQL Boolean Expressions −

```
SQL> SELECT * FROM CUSTOMERS WHERE SALARY =10000;

+----+-------+-----+---------+----------+
| ID | NAME  | AGE | ADDRESS | SALARY   |
+----+-------+-----+---------+----------+
|7|Muffy|24|Indore|10000.00|
+----+-------+-----+---------+----------+
1 row inset(0.00 sec)
```

2. Numeric Expression

These expressions are used to perform any mathematical operation in any query. Following is the syntax −

```
SELECT numerical_expression as  OPERATION_NAME
[FROM table_name
WHERE CONDITION] ;
```

Here, the numerical_expression is used for a mathematical expression or any formula. Following is a simple example showing the usage of SQL Numeric Expressions −

```
SQL> SELECT (15+6) AS ADDITION

+----------+
| ADDITION |
+----------+
|21|
+----------+
1 row inset(0.00 sec)
```

There are several built-in functions like avg(), sum(), count(), etc., to perform what is known as the aggregate data calculations against a table or a specific table column.

```
SQL> SELECT COUNT(*) AS "RECORDS" FROM CUSTOMERS;
```

```
+---------+
| RECORDS |
+---------+
|7|
+---------+
1 row inset(0.00 sec)
```

3. Date Expressions

Date Expressions return current system date and time values −

```
SQL> SELECT CURRENT_TIMESTAMP;
+---------------------+
|Current_Timestamp|
+---------------------+
|2009-11-1206:40:23|
+---------------------+
1 row inset(0.00 sec)
```

Another date expression is as shown below −

```
SQL> SELECT GETDATE();;
+-------------------------+
| GETDATE                 |
+-------------------------+
|2009-10-2212:07:18.140|
+-------------------------+
1 row inset(0.00 sec)
```

Activity

Trainer will ask participants to implement the below questions in your lab sessions based on the above studied concepts.

1. Write a query in SQL to display a table containing employee details as shown in the picture below:

emp_id	emp_name	job_name	manager_id	hire_date	salary	commission	dep_id
68319	KAYLING	PRESIDENT		1991-11-18	6000.00		1001
66928	BLAZE	MANAGER	68319	1991-05-01	2750.00		3001
67832	CLARE	MANAGER	68319	1991-06-09	2550.00		1001
65646	JONAS	MANAGER	68319	1991-04-02	2957.00		2001
67858	SCARLET	ANALYST	65646	1997-04-19	3100.00		2001
69062	FRANK	ANALYST	65646	1991-12-03	3100.00		2001

   Here, students use CREATE and INSERT command to create the table and insert the values.
2. Use SELECT command to display the table created above.
3. Use ALTER and then DROP command to drop the column “commission”.
4. Use ALTER command and then ADD command to add column “commission” back.
5. Use UPDATE column to add some values to the column “commission” where dep_id=1001 OR dep_id=3001.
6. Use DELETE command to delete rows where job_name=”ANALYST”.
7. Use DELETE command to delete rows where job_name=”MANAGER” and emp_name=”CLARE”.
8. Use UPDATE command to add manager_id as 68310 where job_name=”PRESIDENT”.
9. Use TRUNCATE command to remove all the rows from the table. And then use SELECT command to check whether the table contents exist or not.
10. Use DROP command to drop the table.

Joins

A JOIN clause is used to combine rows from two or more tables, based on a related column between them.Different Types of SQL JOINs are:

- **(INNER) JOIN**: Returns records that have matching values in both tables
- **LEFT (OUTER) JOIN**: Return all records from the left table, and the matched records from the right table
- **RIGHT (OUTER) JOIN**: Return all records from the right table, and the matched records from the left table
- **FULL (OUTER) JOIN**: Return all records when there is a match in either left or right table

Example: Consider the two tables below:

Student

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	HARSH	DELHI	8759770477	18
2	PRATIK	BIHAR	7333834034	19
3	RIYANKA	SILIGURI	9876543210	20
4	DEEP	RAMNAGAR	8520369741	18
5	SAPTARHI	KOLKATA	9654783210	19
6	DHANRAJ	BARABAJAR	7412589630	20
7	ROHIT	BALURGHAT	9630258741	18
8	NIRAJ	ALIPUR	7412356890	19

StudentCourse

COURSE_ID	ROLL_NO
1	1
2	2
2	3
3	4
1	5
4	9
5	10
4	11

1. **INNER JOIN**: The INNER JOIN keyword selects all rows from both the tables as long as the condition satisfies. This keyword will create the result-set by combining all rows from both the tables where the condition satisfies i.e value of the common field will be same.

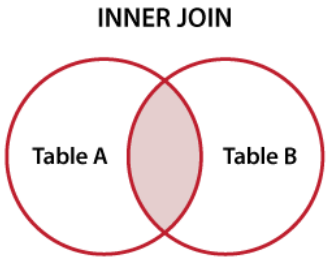
   Syntax:

```
SELECT table1.column1,table1.column2,table2.column1,....
FROM table1
INNER JOIN table2
ON table1.matching_column = table2.matching_column;
```

table1: First table.

table2: Second table
matching_column: Column common to both the tables.

Tip: We can also write JOIN instead of INNER JOIN. JOIN is same as INNER JOIN.



Example Queries(INNER JOIN)

This query will show the names and age of students enrolled in different courses.

```
SELECT StudentCourse.COURSE_ID, Student.NAME, Student.AGE FROM Student
INNER JOIN StudentCourse
ON Student.ROLL_NO = StudentCourse.ROLL_NO;
```

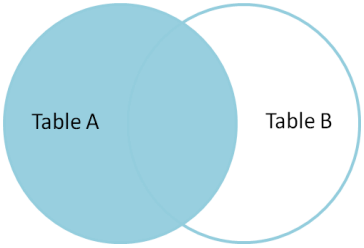
Output:

COURSE_ID	NAME	Age
1	HARSH	18
2	PRATIK	19
2	RIYANKA	20
3	DEEP	18
1	SAPTARHI	19

2. **LEFT JOIN**: This join returns all the rows of the table on the left side of the join and matching rows for the table on the right side of join. The rows for which there is no matching row on right side, the result-set will contain null. LEFT JOIN is also known as LEFT OUTER JOIN.
   Syntax:

```
SELECT table1.column1,table1.column2,table2.column1,....

FROM table1

LEFT JOIN table2

ON table1.matching_column = table2.matching_column;

table1: First table.

table2: Second table

matching_column: Column common to both the tables.
```

Tip: We can also use LEFT OUTER JOIN instead of LEFT JOIN, both are same.



Example Queries(LEFT JOIN):

```
SELECT Student.NAME,StudentCourse.COURSE_ID

FROM Student

LEFT JOIN StudentCourse

ON StudentCourse.ROLL_NO = Student.ROLL_NO;
```

Output:

NAME	COURSE_ID
HARSH	1
PRATIK	2
RIYANKA	2
DEEP	3
SAPTARHI	1
DHANRAJ	*NULL*
ROHIT	*NULL*
NIRAJ	*NULL*

3. **RIGHT JOIN**: RIGHT JOIN is similar to LEFT JOIN. This join returns all the rows of the table on the right side of the join and matching rows for the table on the left side of join. The rows for which there is no matching row on left side, the result-set will contain null. RIGHT JOIN is also known as RIGHT OUTER JOIN.

   Syntax:

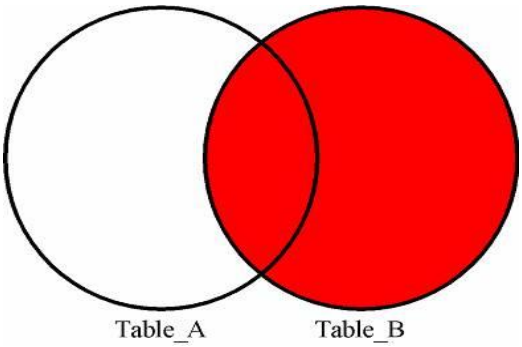
```
SELECT table1.column1,table1.column2,table2.column1,....
FROM table1
RIGHT JOIN table2
ON table1.matching_column = table2.matching_column;
```

table1: First table.

table2: Second table

matching_column: Column common to both the tables.

Tip: We can also use RIGHT OUTER JOIN instead of RIGHT JOIN, both are same.



Example Queries(RIGHT JOIN):

```
SELECT Student.NAME,StudentCourse.COURSE_ID
FROM Student
RIGHT JOIN StudentCourse
ON StudentCourse.ROLL_NO = Student.ROLL_NO;
```

Output:

NAME	COURSE_ID
HARSH	1
PRATIK	2
RIYANKA	2
DEEP	3
SAPTARHI	1
NULL	4
NULL	5
NULL	4

4. **FULL JOIN:** FULL JOIN creates the result-set by combining result of both LEFT JOIN and RIGHT JOIN. The result-set will contain all the rows from both the tables. The rows for which there is no matching, the result-set will contain NULL values.

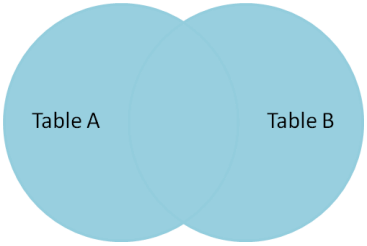
Syntax:

```
SELECT table1.column1,table1.column2,table2.column1,....
FROM table1
FULL JOIN table2
ON table1.matching_column = table2.matching_column;
```

table1: First table.

table2: Second table

matching_column: Column common to both the tables.



Example Queries(FULL JOIN):

```
SELECT Student.NAME,StudentCourse.COURSE_ID
FROM Student
FULL JOIN StudentCourse
ON StudentCourse.ROLL_NO = Student.ROLL_NO;
```

Output:

NAME	COURSE_ID
HARSH	1
PRATIK	2
RIYANKA	2
DEEP	3
SAPTARHI	1
DHANRAJ	*NULL*
ROHIT	*NULL*
NIRAJ	*NULL*
NULL	9
NULL	10
NULL	11

SQL | Join (Cartesian Join & Self Join)

- CARTESIAN JOIN
- SELF JOIN

Consider the two tables below:

Student

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	Ram	Delhi	9455123451	18
2	RAMESH	GURGAON	9652431543	18
3	SUJIT	ROHTAK	9156253131	20
4	SURESH	Delhi	9156768971	18

StudentCourse

COURSE_ID	ROLL_NO
1	1
2	2
2	3
3	4

1. **CARTESIAN JOIN:**

The CARTESIAN JOIN is also known as CROSS JOIN. In a CARTESIAN JOIN there is a join for each row of one table to every row of another table. This usually happens when the matching column or WHERE condition is not specified.

- In the absence of a WHERE condition the CARTESIAN JOIN will behave like a CARTESIAN PRODUCT. i.e., the number of rows in the result-set is the product of the number of rows of the two tables.
- In the presence of WHERE condition this JOIN will function like a INNER JOIN.
- Generally speaking, Cross join is similar to an inner join where the join-condition will always evaluate to True

Syntax:

```
SELECT table1.column1, table1.column2, table2.column1...

FROM table1

CROSS JOIN table2;



table1: First table.
table2: Second table
```

Example Queries (CARTESIAN JOIN):

In the below query we will select NAME and Age from Student table and COURSE_ID from StudentCourse table. In the output you can see that each row of the table Student is joined with every row of the table StudentCourse. The total rows in the result-set = 4 * 4 = 16.

```
SELECT Student.NAME, Student.AGE, StudentCourse.COURSE_ID

FROM Student

CROSS JOIN StudentCourse;
```

Output:

NAME	AGE	COURSE_ID
Ram	18	1
Ram	18	2
Ram	18	2
Ram	18	3
RAMESH	18	1
RAMESH	18	2
RAMESH	18	2
RAMESH	18	3
SUJIT	20	1
SUJIT	20	2
SUJIT	20	2
SUJIT	20	3
SURESH	18	1
SURESH	18	2
SURESH	18	2
SURESH	18	3

2. **SELF JOIN**:

As the name signifies, in SELF JOIN a table is joined to itself. That is, each row of the table is joined with itself and all other rows depending on some conditions. In other words, we can say that it is a join between two copies of the same table.

Syntax:

```
SELECT a.coulmn1 , b.column2

FROM table_name a, table_name b
```

```
WHERE some_condition;
```

table_name: Name of the table.
some_condition: Condition for selecting the rows.

Example Queries(SELF JOIN):

```
SELECT a.ROLL_NO , b.NAME
FROM Student a, Student b
WHERE a.ROLL_NO < b.ROLL_NO;
```

Output:

ROLL_NO	NAME
1	RAMESH
1	SUJIT
2	SUJIT
1	SURESH
2	SURESH
3	SURESH

Executing Queries

Statement objects allow you to execute basic SQL queries and retrieve the results through the *ResultSet* class, which is described later.

To create a Statement instance, you call the *createStatement()* method on the Connection object you have retrieved using one of the *DriverManager.getConnection()* or *DataSource.getConnection()* methods described earlier.

Once you have a Statement instance, you can execute a SELECT query by calling the *executeQuery(String)* method with the SQL you want to use.

To update data in the database, use the *executeUpdate(String SQL)* method. This method returns the number of rows matched by the update statement, not the number of rows that were modified.

If you do not know ahead of time whether the SQL statement will be a SELECT or an UPDATE/INSERT, then you can use the execute(String SQL) method. This method will return true if the SQL query was a SELECT, or false if it was an UPDATE, INSERT, or DELETE statement. If the statement was a SELECT query, you can retrieve the results by calling the getResultSet() method. If the statement was an UPDATE, INSERT, or DELETE statement, you can retrieve the affected rows count by calling *getUpdateCount()* on the Statement instance.

Let's see how to execute a SELECT query using java.sql.Statement:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;
import java.sql.ResultSet;

// assume that conn is an already created JDBC connection (see previous examples)

Statement stmt = null;
ResultSet rs = null;
try {
   stmt = conn.createStatement();
   rs = stmt.executeQuery("SELECT foo FROM bar");

   // or alternatively, if you don't know ahead of time that
   // the query will be a SELECT...

   if (stmt.execute("SELECT foo FROM bar")) {
      rs = stmt.getResultSet();
```

```
    }

    // Now do something with the ResultSet ....
}
catch (SQLException ex){
    // handle any errors
    System.out.println("SQLException: " + ex.getMessage());
    System.out.println("SQLState: " + ex.getSQLState());
    System.out.println("VendorError: " + ex.getErrorCode());

}
finally {
    // it is a good idea to release
    // resources in a finally{} block
    // in reverse-order of their creation
    // if they are no-longer needed

    if (rs != null) {
        try {
            rs.close();
        } catch (SQLException sqlEx) { } // ignore
        rs = null;
    }

    if (stmt != null) {
        try {

            stmt.close();
        } catch (SQLException sqlEx) { } // ignore

        stmt = null;
    }
}
```

Now similarly let us understand how to execute any type of query in JDBC:

```
package com.java2novice.jdbc;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class MyExecuteMethod {

   public static void main(String a[]){

      Connection con = null;
      try {
         Class.forName("oracle.jdbc.driver.OracleDriver");
         con = DriverManager.
               getConnection("jdbc:oracle:thin:@<hostname>:<port num>:<DB name>"
                  ,"user","password");
         Statement stmt = con.createStatement();
         //The query can be update query or can be select query
         String query = "select * from emp";
         boolean status = stmt.execute(query);
         if(status){
            //query is a select query.
            ResultSet rs = stmt.getResultSet();
            while(rs.next()){
               System.out.println(rs.getString(1));
            }  rs.close();
         } else {
            //query can be update or any query apart from select query
            int count = stmt.getUpdateCount();
            System.out.println("Total records updated: "+count);
         }
      } catch (ClassNotFoundException e) {
         // TODO Auto-generated catch block
         e.printStackTrace();
      } catch (SQLException e) {
         // TODO Auto-generated catch block
         e.printStackTrace();
      } finally{
         try{
```

```
            if(con != null) con.close();
          } catch (Exception ex){}
       }
    }
}
```