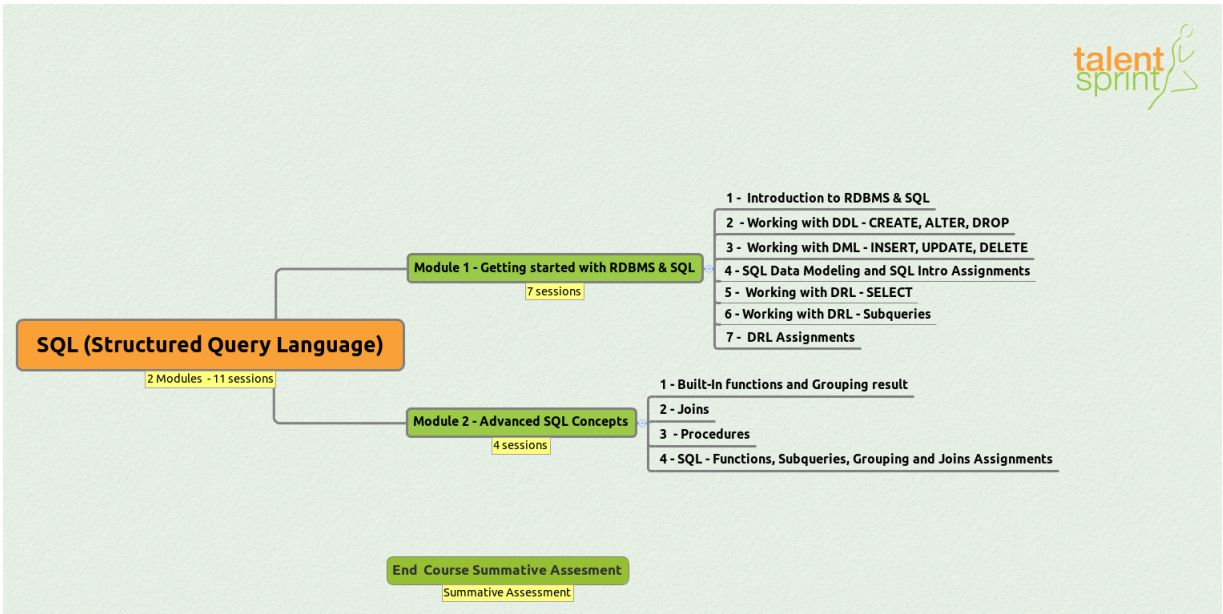
SQL
(Structured Query Language)



Module 1 - Getting started with RDBMS & SQL

Session 1 - Introduction to RDBMS & SQL

Session 2 - Working with DDL - CREATE, ALTER, DROP

Session 3 - Working with DML - INSERT, UPDATE, DELETE

Session 4 - SQL Data Modeling and SQL Intro Assignments (Practice Session)

Session 5 - Working with DRL - SELECT

Session 6 - Working with DRL - Subqueries

Session 7 - DRL Assignments (Practice Session)

Module 2 - Advanced SQL Concepts

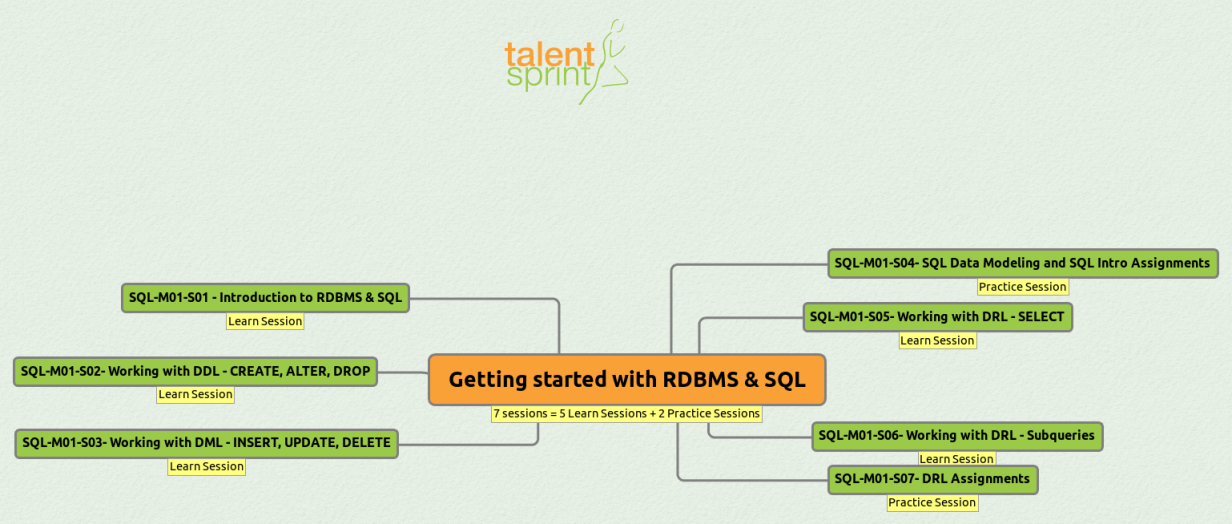
Session 1 - Built-In functions and Grouping result

Session 2 - Joins

Session 3 - Procedures

Session 4 - SQL - Functions, Subqueries, Grouping and Joins Assignments (Practice Session)

Module 1 - Getting started with RDBMS & SQL



Session 1 - Introduction to RDBMS & SQL

What is Database

To find out what database is, we have to start from data, which is the basic building block of any DBMS.

Data Facts, figures, statistics etc. having no particular meaning (e.g. 1, Sachin, 35 etc).

Record Collection of related data items, e.g. in the above example the three data items had no meaning. But if we organize them in the following way, then they collectively represent meaningful information.

Roll No	Name	Age
1	Sachin	35

Table or Relation

Collection of related records.

Roll No	Name	Age
1	Sachin	35
2	Saurav	32
3	Sehwag	35

The columns of this relation are called **Fields**, **Attributes** or **Domains**. The rows are called **Tuples** or **Records**.

Database

Collection of related relations. Consider the following collection of tables:

Table 1

Roll No	Name	Age
1	Sachin	35
2	Saurav	32
3	Sehwag	35

Table 2

Rollno	Address
1	Mumbai
2	Kolkata
3	Delhi

Table 3

Rollno	Year
1	III
2	II
3	I

Table 4

Rollno	Hostel
1	H1
2	H2
3	H3

We now have a collection of 4 tables. They can be called a "related collection" because we can clearly find out that there are some common attributes existing in a selected pair of tables. Because of these common attributes we may combine the data of two or more tables together to find out the complete details of a student. Questions like "Which hostel does the youngest student live in?" can be answered now, although **Age** and **Hostel** attributes are in different tables.

In a database, data is organized strictly in row and column format. The rows are called **Tuple** or **Record**. The data items within one row may belong to different data types. On the other hand, the columns are often called Domain or Attribute. All the data items within a single attribute are of the same data type.

What is Database Management System

A management system is a set of rules and procedures which help us to create organize and manipulate the database. It also helps us to add, modify

delete data items in the database.
The management system can be either manual or computerized.

The management system is important because without the existence of some kind of rules and regulations it is not possible to maintain the database. We have to select the particular attributes which should be included in a particular table; the common attributes to create relationship between two tables; if a new record has to be inserted or deleted then which tables should have to be handled etc. These issues must be resolved by having some kind of rules to follow in order to maintain the integrity of the database.

Relational Database Management System

A DBMS that is based on relational model is called as RDBMS. Relation model is most successful mode of all three models. Designed by E.F. Codd, relational model is based on the theory of sets and relations of mathematics.

Relational model represents data in the form a table. A table is a two dimensional array containing rows and columns. Each row contains data related to an entity such as a student. Each column contains the data related to a single attribute of the entity such as student name.

Terminologies

Tuple/Row

A single row in the table is called as tuple. Each row represents the data of a single entity.

Attribute/Column

A column stores an attribute of the entity. For example, if details of students are stored then student name is an attribute; course is another attribute and so on.

Primary Key

If you take STUDETNS table, it contains data related to students. For each student there will be one row in the table.Each students data in the table must be uniquely identified. In order to identify each entity uniquely in the table, we use a column in the table. That column, which is used to uniquely identify entities (students) in the table, is called as **Primary Key**.

Composite Primary Key

In some tables a single column cannot be used to uniquely identify entities (rows). In that case we have to use two or more columns to uniquely identify rows of the table. When a primary key contains two or more columns it is called as **Composite Primary Key**.

Foreign Key

In relational model, we often store data in different tables and put them together to get complete information. For example, in PAYMENTS table we have only ROLLNO of the student. To get remaining information about the student we have to use STUDETNS table. Roll number in PAYMENTS table can be used to obtain remaining information about the student.

The relationship between entities student and payment is one-to-many. One student may make payment for many times. As we already have ROLLNO column in PAYMENTS table, it is possible to join with STUDENTS table and get information about parent entity (student). Roll number column of PAYMENTS table is called as foreign key as it is used to join PAYMENTS table with STUDENTS table. So, **Foreign Key** is the key on the many side of the relationship.

When a child table contains a row that doesnt refer to a corresponding parent key, it is called as **Orphan Record**. We must not have orphan records, as they are result of lack of data integrity.

Structured Query Language

Almost all relational database management systems use SQL (Structured Query Language) for data manipulation and retrieval.

SQL is the standard language for relational database systems.SQL is a non-procedural language, where you need to concentrate on what you want, not on how you get it.

SQL Commands are divided into following categories, depending upon what they do.

DDL (Data Definition Language): DDL commands are used to define, modify and delete the structure of database and its objects. Example: **CREATE**, **ALTER**, **DROP** etc.

DML (Data Manipulation Language): DML commands are used to insert, update and delete data to existing table. Example: **INSERT**, **UPDATE**, **DELETE**

DCL (Data Control Language): DCL commands are used to grant and revoke different privilege to/from user. Example: **GRANT**, **REVOKE**

TCL (Transaction Control Language): TCL commands are used to complete a transaction. Example: **COMMIT**, **ROLLBACK**

DQL/DRL (Data Query/Retrieving Language): DQL/DRL command is used to select retrieve data from database. Example: **SELECT**

Advantages of Database Management System

We must evaluate whether there is any gain in using a DBMS over a situation where we do not use it. Let us summarize the advantages.

Reduction of Redundancy: This is perhaps the most significant advantage of using DBMS. Redundancy is the problem of storing the same data item in more than one place. Redundancy creates several problems like requiring extra storage space, entering same data more than once during data insertion, and deleting data from more than one place during deletion.

Sharing of Data: In a paper-based record keeping, data cannot be shared among many users. But in computerized DBMS, many users can share the same database if they are connected via a network.

Data Integrity: We can maintain data integrity by specifying integrity constrains, which are rules and restrictions about what kind of data may be entered or manipulated within the database. This increases the reliability of the database as it can be guaranteed that no wrong data can exist within the database at any point of time.

Data security: We can restrict certain people from accessing the database or allow them to see certain portion of the database while blocking sensitive information. This is not possible very easily in a paper-based record keeping.

Data Independence: Data independence means that "the application is independent of the storage structure and access strategy of data". In other words, the ability to modify the schema definition in one level should not affect the schema definition in the next higher level. There are two types of Data Independence:

Physical Data Independence: Modification in physical level should not affect the logical level.

Logical Data Independence: Modification in logical level should not affect the view level.

Data Abstraction: It means hiding implementation details (i.e. high level details) from end user. e.g. In case of storage of data in database user can only access the database, but implementation details such as how the data is stored physically onto the disc is hidden from user. There three levels of

abstraction:

Physical level: The lowest level of abstraction describes how data are stored.

Logical level: The next higher level of abstraction, describes what data are stored in database and what relationship among those data.

View level: The highest level of abstraction describes only part of entire database.

Constraints

Data Integrity

SQL Constraints are rules used to limit the type of data that can go into a table, to maintain the accuracy and integrity of the data inside table. Constraints are of two types:

- Column level Constraints:limits only column data.
- Table level Constraints: applicable to whole table.

Following are the constraints that can be applied to table.

NOT NULL Constraint:

The NOT NULL constraint enforces a column NOT to accept NULL values. NOT NULL can't be applied to table level.

Example:
mysql¿ CREATE TABLE Persons(PID int NOT NULL, LastName varchar(30) NOT NULL, FirstName varchar(30), Address varchar(60));

UNIQUE Constraint:

The UNIQUE constraint uniquely identifies each record in a database table.Unique constraint can be applied to column level or table level.

Example:
mysql¿CREATE TABLE Persons(PID int ,LastName varchar(30),FirstName varchar(30),Address varchar(70),UNIQUE (PID)); NOTE: Above constraint is applied on table level.

Defining UNIQUE Constraint on ALTER TABLE:

To create a UNIQUE constraint on the "PID" column when the table is already created.

Example:
mysql¿ ALTER TABLE Persons ADD UNIQUE (PID);

Naming a Constraint:

To allow naming of a UNIQUE constraint, and for defining a UNIQUE constraint on multiple columns

Example:
mysql¿ ALTER TABLE Persons ADD CONSTRAINT uc_PersonID UNIQUE (PID,LastName);
NOTE:uc_PersonID is the Constraint name.

Dropping a Constraint:

To drop a UNIQUE constraint, use the following command:

Example:
mysql¿ ALTER TABLE Persons DROP INDEX uc_PersonID;

PRIMARY KEY Constraint:

A Primary Key uniquely identifies each record. A Primary Key must contain unique values and it must not contain null value.

Example:
mysql¿ CREATE TABLE Persons(PID int ,LastName varchar(30),FirstName varchar(30),Address varchar(70),PRIMARY KEY (PID));

Defining Primary Key Constraint on ALTER TABLE:

To create a Primary Key constraint on the "PID" column when the table is already created.

Example:
mysql¿ ALTER TABLE Persons ADD Primary Key (PID);

Naming a Constraint:

To allow naming of a UNIQUE constraint, and for defining a UNIQUE constraint on multiple columns

Example:
mysql¿ ALTER TABLE Persons ADD CONSTRAINT uc_PersonID Primary Key (PID,LastName);
NOTE:uc_PersonID is the Constraint name.

Dropping a Constraint:

To drop a Primary Key constraint, use the following command:

Example:
mysql¿ ALTER TABLE Persons DROP Primary Key;

FOREIGN KEY Constraint:

A FOREIGN KEY in one table points to a PRIMARY KEY in another table.

Illustration:

Let's illustrate the foreign key with an example. Look at the following two tables:

Person Table

PID	Name	City
1111	Sachin	Mumbai
1112	Ganguly	Kolkata

Orders Table

OID	OrderNo	PID
1	1234	1111
2	1235	1111

Note:
That the "PID" column in the "Orders" table points to the "PID" column

in the “Persons” table.
The “PID” column in the “Persons” table is the PRIMARY KEY in the “Persons” table.
The “PID” column in the “Orders” table is a FOREIGN KEY in the “Orders” table.
The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables.
The FOREIGN KEY constraint also prevents that invalid data from being inserted into the foreign key column, because it has to be one of the values contained in the table it points to.

The following SQL creates a FOREIGN KEY on the “PID” column when the “Orders” table is created:
Example:
mysql> CREATE TABLE Orders(OId int ,OrderNo int ,PId int,PRIMARY KEY (OId),FOREIGN KEY (PId) REFERENCES Persons(PId));

DEFAULT Constraint:

The DEFAULT constraint is used to insert a default value into a column. The default value will be added to all new records, if no other value is specified.

Example:
mysql> CREATE TABLE Persons(PID int ,LastName varchar(25),FirstName varchar(25),Address varchar(55),City varchar(55) DEFAULT ’Sandnes’);

To create a DEFAULT constraint on the “City” column when the table is already created,use the following syntax:

mysql> ALTER TABLE Persons ALTER City SET DEFAULT ’SANDNES’;

To drop a DEFAULT constraint, use the following syntax:

mysql> ALTER TABLE Persons ALTER City DROP DEFAULT;

Session 2 - Working with DDL - CREATE, ALTER, DROP

How to connect to MySql

$ mysql u root p
Enter password: root

How to display available Databases

mysql>SHOW databases;

How to create a new Database

mysql> CREATE DATABASE demo;

How to connect to a particular Database

mysql> USE employee;

mysql> CONNECT employee;

How to know which Database user is connected

mysql>SELECT DATABASE() FROM DUAL;

How to display all tables available in connected Database

mysql>SHOW TABLES;

Creating a Table

mysql> CREATE TABLE employee (empid INT, ename VARCHAR (20), salary FLOAT (10, 2));

How to create a table by copying the structure and data of an existing Table

mysql> CREATE TABLE student AS SELECT * FROM employees;

How to create a table by copying the structure only not the data of an existing Table

mysql> CREATE TABLE student AS SELECT * FROM employees where 1=2;

How to describe the structure of a Table

mysql>DESC employee;

Alter

How to add a new column to existing Table

mysql>ALTER TABLE employee add address varchar(20);
mysql>ALTER TABLE employee add address varchar(20),add phone int;

How to modify structure of exisiting column of a Table

mysql> ALTER TABLE employee MODIFY address VARCHAR (40);
mysql> ALTER TABLE employee MODIFY ename VARCHAR (22), MODIFY city VARCHAR (22);

How to drop exisiting column from a Table

mysql> ALTER TABLE employee DROP phone;
mysql> ALTER TABLE employee DROP ename, DROP salary;

How to rename existing column of a Table

mysql> ALTER TABLE employee CHANGE address newaddress VARCHAR(30) ;

mysql> ALTER TABLE employee CHANGE empid id VARCHAR (100), CHANGE newaddress address VARCHAR (100);

How to rename an exisiting Table

mysql> RENAME TABLE employee TO newemployee;
mysql> RENAME TABLE newemployee TO employees,student TO students, department TO departments;

Drop

How to drop existing Table

mysql>DROP TABLE employee;

How to delete all records of a table permanently

mysql>TRUNCATE employee;

Data Types

- **SMALLINT**: A small integer. The signed range is -32768 to 32767. The unsigned range is 0 to 65535.
- **MEDIUMINT**: A medium-sized integer. The signed range is -8388608 to 8388607. The unsigned range is 0 to 16777215.
- **INT**: A normal-size integer. The signed range is -2147483648 to 2147483647. The unsigned range is 0 to 4294967295.
- **BIGINT**: A large integer. The signed range is -9223372036854775808 to 9223372036854775807. The unsigned range is 0 to 18446744073709551615.
- **FLOAT**: A small (single-precision) floating-point number. Permissible values are -3.402823466E+38 to -1.175494351E-38, 0, and 1.175494351E-38 to 3.402823466E+38. These are the theoretical limits, based on the IEEE standard. The actual range might be slightly smaller depending on your hardware or operating system.
- **DATE**: A date. The supported range is '1000-01-01' to '9999-12-31'. MySQL displays DATE values in 'YYYY-MM-DD' format.
- **TIME** : A time. The range is '-838:59:59' to '838:59:59'. MySQL displays TIME values in 'HH:MM:SS' format.

- **DATETIME**: A date and time combination. The supported range is '1000-01-01 00:00:00' to '9999-12-31 23:59:59'. MySQL displays DATETIME values in 'YYYYMM-DD HH:MM:SS' format.

- **YEAR[(2—4)]**: A year in two-digit or four-digit format. The default is four-digit format.In four-digit format, the permissible values are 1901 to 2155, and 0000. In two-digit format, the permissible values are 70 to 69, representing years from 1970 to 2069.MySQL displays YEAR values in YYYY format.

- **TIMESTAMP**: A timestamp. The range is '1970-01-01 00:00:01' UTC to '2038-01-19 03:14:07' UTC.

- **CHAR** :The length of a CHAR column is fixed to the length that you declare when you create the table. The length can be any value from 0 to 255.

- **VARCHAR**: The length can be specified as a value from 0 to 255 before MySQL 5.0.3, and 0 to 65,535 in 5.0.3 and later versions.

Session 3 - Working with DML - INSERT, UPDATE, DELETE

Data Manipulation Language (DML) statements are used for managing data within schema objects (e.g. table).

It includes Inserting, Updating and deleting of data.

Insert Statement

INSERT statement allows you to insert one or more rows to the table.It is possible to write INSERT INTO in two forms:

The first form doesn't specify the column names where the data will be inserted, only their values. It is used where users need to insert values to all available columns of a table.

Syntax
INSERT INTO tablename VALUES (value1, value2, value3,...);

Example
mysql> INSERT INTO student VALUES (1,'Ajay','Hyderabad');

The second form specifies both the column names and the values to be inserted. It is used where users need to insert value to specific column(s) of a given table.

Syntax
INSERT INTO tablename (column1, column2,column3,...) VALUES (value1, value2, value3,...)

Example
mysql> INSERT INTO student (Rollno, Name)VALUES (2, 'Rahul');

Update Statement

SQL UPDATE statement is used to update existing data in database tables. It can be used to change values of single row, group of rows or even all rows in a table.

The SET clause determines the column(s) name and the changed value(s). The changed values could be a constant value, expression or even a subquery.

WHERE clause determines which rows of the tables will be updated. It is an optional part of SQL UPDATE statement. If WHERE clause is ignored, all rows in the tables will be updated.

In employees table, if you want to update the email of Mary with employeeid 1 with the new email as mson@talentsprint.com, you can execute the following query.

Syntax UPDATE tablename SET columnname=value [where condition];
Example mysql> UPDATE employees SET email ='mson@talentsprint.com' WHERE employeeid = 1;

The following syntax will update the email of all employees with the new email as mson@talentsprint.com.

mysql> UPDATE employees SET email ='mson@talentsprint.com';

Delete Statement

The DELETE FROM statement is used to delete records from a database table. The WHERE clause in DELETE statement specifies condition to limit which rows you want to remove.

Syntax DELETE from tablename [where condition];
Example DELETE from employees where salary=10000;

The following syntax will delete all records from employees.

Syntax DELETE from tablename;
Example DELETE from employees;

Session 4 - SQL Data Modeling and SQL Intro Assignments (Practice Session)

Session 5 - Working with DRL - SELECT

The SELECT statement is used to retrieve data from a table. SELECT statement has many optional elements that we can use. The order of FROM, WHERE, GROUP BY, HAVING, ORDER BY and LIMIT has to be in the sequence mentioned. To select all columns in a table you can use asterisk (*) notation instead of listing all column names in the SELECT statement. For example, if you need to query all the columns in "employees" table, you can use the following query:
mysql> SELECT * FROM employees;
SELECT statement also allows you to view partial data of a table by listing columns' name after the SELECT keyword. This is called **Projection**.

For example if you need to view only "firstname", "lastname" and "jobtitle" of employee in the "employees" table, you can use the following query:
mysql> SELECT firstname, lastname, jobtitle FROM employees;

Where Clause

WHERE clause of the SELECT statement enables you to select particular rows which match its conditions or search criteria. You use WHERE clause to filter the records based on a certain conditions. For example, you can find the details of president of company by using the following query:

mysql> SELECT firstname, lastname, email FROM employees WHERE jobtitle="president";

Distinct Clause

With DISTINCT clause, you can eliminate the duplicate records while displaying the result. For example, to find how many job titles of all employees in the employees table, you use DISTINCT keyword in SELECT statement as follows:
mysql> SELECT DISTINCT jobtitle FROM employees; mysql> SELECT

DISTINCT (jobtitle) FROM employees;

The DISTINCT clause can be applied with more than one column. In this case, the combinations of all columns are used to define the uniqueness of a record in the return result set.

For example, to get all cities and states of customers in the customers table, we can use the following query:

mysql< SELECT DISTINCT city, state FROM customers;

ORDER BY Clause

The ORDER BY clause allows you to sort the result set on one or more columns in ascending or descending order. To sort the result set in ascending order you use ASC and in descending order you use DESC keywords. By default, the ORDER BY will sort the result set in ascending order. For example, to sort the name of employees by firstname, you can execute the following query:
mysql>SELECT firstname, lastname, jobtitle FROM employees ORDER BY firstname;

LIMIT Clause

MySQL supports a feature called LIMIT to allow you to constrain the returned records with SELECT statement.

Let's say you have a database table with 10000 records and you want to get just first N records, you can use the following query:

Syntax
mysql> SELECT * FROM table LIMIT N;

Example
If you want to get the first five employees in the table employees, you can use the following query:

mysql> SELECT firstname, lastname FROM employees LIMIT 5;

The MySQL LIMIT also allows you to get a range of records where you decide starting record number and how many records you want to retrieve.

Syntax

mysql> SELECT columns FROM table LIMIT S, N;
NOTE: In the query above, S is the starting record index. MySQL specifies that the first record starts with 0. N is the number of records you want to select.

Example
Now if you want to get five employees from employee number 10 you can use MySQL LIMIT with offset as follows:
mysql> SELECT firstname, lastname FROM employees LIMIT 10, 5;

IN Operator

SQL IN allows you to select values which match any one of a list of values. The usage of SQL IN is as follows:
Syntax
SELECT columnlist FROM tablename WHERE column IN ("listitem1","listitem2");

Example
Suppose if you want to find out all offices which are located in US and France, you can perform the following query:

mysql> SELECT officeCode, city, phone FROM offices WHERE country IN ('USA', 'France');

Example
To get all countries which are not located in USA or France, we can use NOT IN in the where clause as follows:

mysql> SELECT officeCode, city, phone FROM offices WHERE country NOT IN ('USA', 'France');

LIKE Operator

MySQL provides LIKE operator in SQL standard. The MySQL LIKE operator is commonly used to select data based on patterns matching.
MySQL provides you two wildcard characters for using with LIKE:

The Percentage (%) wildcard allows you to match any string of zero or more characters.
Underscore (_) allows you to match any single character.

Syntax
SELECT * from table where column LIKE PATTERN;

Example
Suppose you want to search for employee in employees table who has first name starting with character ‘a’, you can do it as follows:

mysql> SELECT * FROM employees WHERE firstname LIKE ‘a%’;

Session 6 - Working with DRL - Subqueries

A subquery is a SQL query nested inside a larger query.

A subquery may occur in :
A SELECT clause
A FROM clause
A WHERE clause

In MySQL subquery can be nested inside a SELECT, INSERT, UPDATE, DELETE, SET, or DO statement or inside another subquery.
A subquery is usually added within the WHERE Clause of another SQL SELECT statement.
You can use the comparison operators, such as >, <, or =. The comparison operator can also be a multiple-row operator, such as IN, ANY, SOME, or ALL.
A subquery can be treated as an inner query, which is a SQL query placed as a part of another query called as outer query.
The inner query executes first before its parent query so that the results of inner query can be passed to the outer query.
Syntax
SELECT select_list from outerTable where expr_operator(SELECT select_list from innerTable);

In the above statement the inner query will be parsed first and the result will be passed to the outer query.

Types of SubQueries

- Scalar SubQueries
- SubQueries with ANY,ALL,IN
- SubQueries with EXISTS and NOT EXISTS

- Correlated SubQueries

Scalar SubQueries

- A scalar subquery is a subquery that returns exactly one column value from one row.
- If the subquery returns 0 rows then the value of scalar subquery expression in NULL.
- If the subquery returns more than one row then MySQL returns an error.
- The subquery can be used in either SELECT statement or WHERE clause.
- In either case, an aggregate function or an expression is normally used in the subquery.
- When the subquery is used in WHERE clause, a comparison operator is always used

Subquery in WHERE clause with an aggregate function

mysql>select OrderID, CustomerID from orders where ShippedDate = (select max(ShippedDate) from orders);

```
mysql> select orderid,customerid from orders where shippeddate=(select  max(shippedd
te) from orders);
+---------+------------+
| orderid | customerid |
+---------+------------+
|   11063 | HUNGO      |
|   11067 | DRACD      |
|   11069 | TORTU      |
+---------+------------+
3 rows in set (0.00 sec)
```

Figure 1: figure
Scalar SubQuery using Max function

NOTE:The above query returns data for all customers and their orders where the orders were shipped on the most recent recorded day.

mysql>select distinct ProductName, UnitPrice from products where UnitPrice>(select avg(UnitPrice) from products) order by Unit-

Price desc;

NOTE:This query returns all products whose unit price is greater than average unit price of all Products.

SubQueries using ALL

The ALL keyword specifies that the search condition is TRUE if the comparison is TRUE for every value that the subquery returns.

In the following example, the first condition tests whether each unitprice is greater than the unitprice of productid=1.

mysql>select productname,unitprice from products where unitprice > ALL (SELECT unitprice FROM products WHERE productid = 1);

```
mysql>
mysql> select productName,unitprice from products where unitprice > ALL (SELECT unitp
rice FROM products WHERE productid = 1);
+---------------------------------+-----------+
| productName                     | unitprice |
+---------------------------------+-----------+
| Chang                           |        19 |
| Chef Anton's Cajun Seasoning    |        22 |
| Chef Anton's Gumbo Mix          |     21.35 |
| Grandma's Boysenberry Spread    |        25 |
| Uncle Bob's Organic Dried Pears |        30 |
| Northwoods Cranberry Sauce      |        40 |
```

Figure 2: figure
Subquery using ALL

Subquery using ANY

The ANY keyword denotes that the search condition is TRUE if the comparison is TRUE for at least one of the values that is returned. If the subquery returns no value, the search condition is FALSE. The SOME keyword is a synonym for ANY.

mysql>select productName,unitprice from products where unitprice < ANY (SELECT unitprice FROM products WHERE productid = 1);

```
mysql> select productName,unitprice from products where unitprice < ANY (SELECT unitp
rice FROM products WHERE productid = 1);
+-----------------------------------+-----------+
| productName                       | unitprice |
+-----------------------------------+-----------+
| Aniseed Syrup                     |        10 |
| Konbu                             |         6 |
| Genen Shouyu                      |      15.5 |
| Pavlova                           |     17.45 |
| Teatime Chocolate Biscuits        |       9.2 |
| Sir Rodney's Scones               |        10 |
```

Figure 3: figure
SubQuery using ANY

Subquery using EXISTS

The MySQL EXISTS condition is used in combination with a subquery and is considered "to be met" if the subquery returns at least one row. It can be used in a SELECT, INSERT, UPDATE, or DELETE statement.
NOTE:SQL statements that use the EXISTS Condition in MySQL are very inefficient since the sub-query is RE-RUN for EVERY row in the outer query's table.

mysql>select * from customers where EXISTS (select * from order_details where customers.customer_id = orders.customer_id);
NOTE:The above example will return all records from the customers table where there is at least one record in the order_details table with the matching customer_id.

```
mysql>
mysql> select customerid,contactname,country from customers where EXISTS (select * fr
om orders where customers.customerid = orders.customerid);
+------------+-------------------------+-------------+
| customerid | contactname             | country     |
+------------+-------------------------+-------------+
| ALFAA      | Maria Anders            | Germany     |
| ANATR      | Ana Trujillo            | Mexico      |
| ANTON      | Antonio Moreno          | Mexico      |
| AROUT      | Thomas Hardy            | UK          |
| BERGS      | Christina Berglund      | Sweden      |
```

Figure 4: figure
Exists Condition

Subquery using NOT EXISTS

select * from customers where NOT EXISTS (select * from order_details where customers.customer_id = orders.customer_id);
NOTE:The above example will return all records from the customers table

```
mysql> select customerid,contactname,country from customers where NOT EXISTS (select
* from orders where customers.customerid = orders.customerid);
+------------+----------------+---------+
| customerid | contactname    | country |
+------------+----------------+---------+
| FISSA      | Diego Roel     | Spain   |
| PARIS      | Marie Bertrand | France  |
+------------+----------------+---------+
2 rows in set (0.01 sec)
```

Figure 5: figure
Not Exists Condition

when there are no records in the order_details table for the given customer_id.

Subquery using EXISTS and INSERT

INSERT into contacts (contact_id, contact_name) select supplier_id, supplier_name from suppliers where EXISTS (select * from orders where suppliers.supplier_id = orders.supplier_id);
NOTE:The above example will insert records into contacts table if supplier_id of suppliers and orders table matches.

Subquery using EXISTS and DELETE

DELETE from suppliers where EXISTS (select * from orders where suppliers.supplier_id = orders.supplier_id);
NOTE:The above example will delete records from suppliers table if supplier_id of suppliers and orders table matches.

Correlated SubQueries

A correlated subquery is a subquery that contains a reference to a table (in the parent query) that also appears in the outer query.
MySQL evaluates from inside to outside.
mysql> select supplierid, productname from products p where unitprice> (select AVG(unitprice) from products where supplierid = p.supplierid);
Displaying all employees whose salary is greater than the average salary of

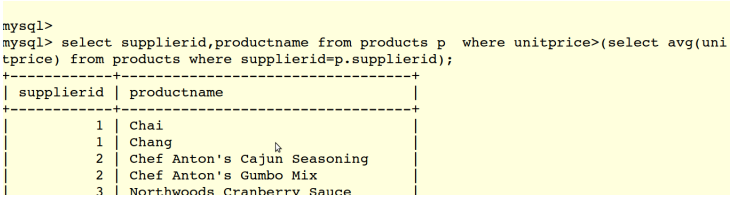
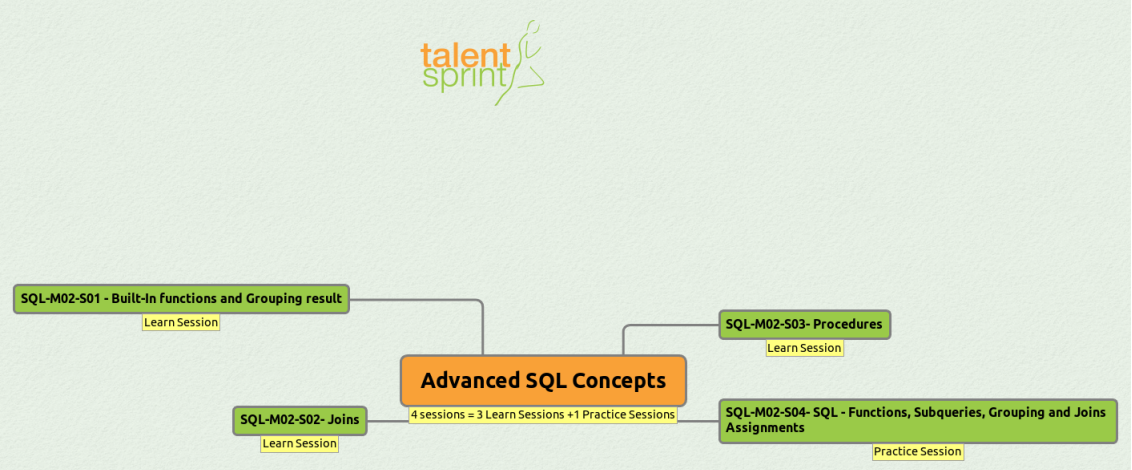


Figure 6: figure
Correlated SubQuery

their own department.

Session 7 - DRL Assignments (Practice Session)

Module 2 - Advanced SQL Concepts



Session 1 - Built-In functions and Grouping result

Functions

MySQL has many built-in functions for performing calculations on data as follows:

Aggregrate Functions

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

AVG()

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

Syntax
SELECT AVG(columnname) FROM tablename;

Example
sql>SELECT AVG(SAL) from employees;

COUNT()

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

textbfSyntax
SELECT COUNT(columnname) FROM tablename;

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

textbfExample
mysql> SELECT COUNT(empno) FROM employees;

MAX()

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

Syntax
SELECT MAX(columnname) FROM tablename;

Example
sql>SELECT MAX(sal) FROM employees;

MIN()

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

Syntax
SELECT MIN(columnname) FROM tablename;

Example
sql>SELECT MIN(sal) FROM employees;

SUM()

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

Syntax
SELECT SUM(columnname) FROM tablename;

Example
sql>SELECT SUM(sal) FROM employees;

ROUND()

: The ROUND() function rounds a numeric field to the number of decimals specified.

Syntax
SELECT ROUND(columnname,number) FROM tablename;

Example
mysql> select ROUND(sal,2) from employees;

TRUNCATE()

The TRUNCATE() truncates a numeric field to the number of decimals specified.

Syntax
SELECT TRUNCATE(columnname,number) FROM tablename;

Example
mysql> SELECT TRUNCATE(sal,2) from employees;

SCALAR Functions

UCASE()

Converts a field to upper case.

Syntax
SELECT UCASE(columnname) FROM tablename;

Example
mysql> SELECT UCASE(ename) from employees;

LCASE()

Converts a field to lower case.

Syntax SELECT LCASE(columnname) FROM tablename;

Example mysql> SELECT LCASE(ename) FROM employees;

LENGTH()

Returns of the length of the text field.

Syntax
SELECT LENGTH(columnname) FROM tablename;

Example
mysql> SELECT LCASE(ename) FROM employees;

MID()

Extracts character from a text field.

Syntax
SELECT MID(column,pos) from tablename;

Example
mysql> SELECT MID(ename,3) FROM emp WHERE ename='SANTOSH';
Note: Above specified 3 is the starting character and ending character.

mysql> SELECT MID(ename,1,3) FROM emp WHERE ename='SANTOSH';
Note: Above specified 1 is the starting character and 3 is the ending character.

REPLACE()

Returns the string with all occurrences of the string from_str replaced by the string to_str. **Syntax** SELECT REPLACE(str,from_str,to_str) FROM tablename;

Example mysql> SELECT REPLACE(ename,'a','A') FROM employee;

DATE Functions

NOW()

Returns the current date and time.

Example
mysql> SELECT NOW() FROM dual;

CURDATE()

Returns the current date.

Example
mysql> SELECT CURDATE() FROM dual;

CURTIME()

Returns the current time.

Example
mysql> SELECT CURTIME() FROM dual;

EXTRACT()

Extracts a single part of a date/time.

Example
mysql> SELECT EXTRACT(year FROM now()) FROM dual;
mysql> SELECT EXTRACT(month FROM now()) FROM dual;
mysql> SELECT EXTRACT(day FROM now()) FROM dual;
mysql> SELECT EXTRACT(hour FROM now()) FROM dual;
mysql> SELECT EXTRACT(minute FROM now()) FROM dual;
mysql> SELECT EXTRACT(second FROM now()) FROM dual;

DATE_ADD()

Adds a specified time interval to a date.

Example
mysql> SELECT DATE_ADD(NOW(),INTERVAL 10 DAY) FROM dual;

DATE_SUB()

Subtracts a specified time interval from a date.

Example
mysql> SELECT DATE_SUB(NOW(),INTERVAL 10 DAY) FROM dual;

DATEDIFF()

Returns the number of days between two dates.

Example
mysql> SELECT DATEDIFF(NOW(),'20130612') FROM dual;
mysql> SELECT DATEDIFF(NOW(),'20000612') FROM dual;
mysql> SELECT (DATEDIFF(NOW(),'20000612'))/365 FROM dual;

DATE_FORMAT()

Specifies the format for date and time.

Formats

Format	Description
%M	Month Name
%m	Month, in Numeric
%D	Day of Month Numeric(th)
%d	Day of Month numberic
%Y	4 Digits

Example mysql¿ SELECT DATE_FORMAT(NOW(),'%d%m%y') FROM dual; mysql¿ SELECT DATE_FORMAT(NOW(),'%D%M%Y') FROM dual

GROUP BY Clause

The MySQL GROUP BY clause is used with SQL SELECT statement to group selected records into a set of summary records by the one or more column's value or expression.
The MySQL GROUP BY clause must appear after the WHERE clause or FROM clause if WHERE clause is omitted of the SQL SELECT statement.

Syntax
SELECT columname(s) from tablename GROUP BY column(s).

Example
Let's say if you want to know how many orders in each status group you can use the COUNT function as follows:

mysql> SELECT status, count (*) FROM orders GROUP BY status;

If you want to see the result of the query above in the descending order, you can do it as follows:

mysql> SELECT status, count (*) FROM orders GROUP BY status DESC;

HAVING Clause

The HAVING clause is an optional part of and used only with the SQL SELECT statement. The HAVING clause specifies a filter condition for a group of record or an aggregate.
The HAVING is often used with GROUP BY clause. When using with GROUP BY clause, you can apply filter condition of the HAVING clause only to the columns appear in the GROUP BY clause.

NOTE
If the GROUP BY clause is omitted, the HAVING clause will behave like a WHERE clause.
HAVING clause applies to groups as a whole while the WHERE clause applies to individual rows.

Example
What order has total value greater than 1000. In this case, you need to use the MySQL HAVING clause on aggregate to answer that question.

mysql> SELECT ordernumber, SUM (price) AS total FROM orderdetails GROUP BY ordernumber HAVING total > 1000;

Session 2 - Joins

Joins are used to query data from two or more tables, based on a relationship between certain columns in these tables. There are different types of JOINing methods used in MySQL database, which are given below.

INNER JOIN

Probably the most common join operation MySQL supports is an inner join. It identifies and combines only matching rows which are stored in two or more related tables.
A join condition, which indicates how the tables are related, is added with the keywords ON or USING

NOTE:

ON is used when the relationship column names are different.

USING is used when the relationship column names are same.

INNER JOIN with ON clause

Syntax
SELECT <column list > FROM tableA a INNER JOIN tableB b ON a.somecolumn = b.othercolumn;

Example
mysql> SELECT d.deptno, d.dname, e.empno, e.ename FROM dept d INNER JOIN emp e ON d.deptno = e.deptid;

INNER JOIN with USING clause syntax

Syntax
SELECT * FROM tableA a INNER JOIN tableB b USING(columnname);

Example
SELECT d.deptno, d.dname, e.empno, e.ename FROM dept d INNER JOIN emp e USING (deptno);

OUTER JOIN

OUTER JOIN allows us to retrieve all values in a certain table regardless of whether these values are present in other tables.

The difference between inner and outer join is: An outer join can identify rows without a match in the joined table. When no match was found, MySQL sets the value of columns from the joined table to NULL.

LEFT OUTER JOIN

This type of join will display matching records from both tables and all unmatched records of the left table. Left table means which table name is listed on the left side of the JOIN keywords.

Syntax
SELECT <column list> FROM table1 a LEFT OUTER JOIN table2 b ON a.somecolumn = b.othercolumn;

Example
SELECT d.deptno, d.dname, e.empno, e.ename FROM dept d LEFT OUTER JOIN employees e ON d.deptno = e.deptnumber;

RIGHT OUTER JOIN

This type of join will display matching records from both tables and all unmatched records of the right table. Right table means which table name is listed on the right side of the JOIN keywords.

Syntax

SELECT <column list> FROM table1 a RIGHT OUTER JOIN table2 b ON a.somecolumn = b.othercolumn;

Example
SELECT d.deptno, d.dname, e.empno, e.ename FROM dept d RIGHT OUTER JOIN employees e ON d.deptno = e.deptnumber;

SELF JOIN

A self-join, also known as an inner join, is a structured query language (SQL) statement where a queried table is joined to itself. The self-join statement is necessary when two sets of data, within the same table, are compared.

Example Suppose you're tasked with writing a SQL query to retrieve a list of employees and their managers.

mysql> SELECT e.firstname AS 'Employee FN', e.lastname AS 'Employee LN', m.firstname AS 'Manager FN', m.lastname AS 'Manager LN' FROM employees AS e LEFT OUTER JOIN employees AS m ON e.manager =m.id;

CROSS JOIN

The cross join operation retrieves data between two tables as a Cartesian product of set theory in mathematics. Each row will get multiplied by other rows. If one table has three rows and the second row has two rows, then the Cartesian of two table will be six.

Example

mysql> SELECT * FROM dept CROSS JOIN employee;

Session 3 - Procedures

0.1 Procedures

A `subprogram` is a program unit/module that performs a particular task. These subprograms are combined to form larger programs. This is basically called the 'Modular design'. A subprogram can be invoked by another subprogram or program which is called the calling program.

A subprogram can be created:

1. At schema level
2. Inside a package
3. Inside a PL/SQL block

A schema level subprogram is a `standalone subprogram`. It is created with the CREATE PROCEDURE or CREATE FUNCTION statement. It is stored in the database and can be deleted with the DROP PROCEDURE or DROP FUNCTION statement.

A subprogram created inside a package is a `packaged subprogram`. It is stored in the database and can be deleted only when the package is deleted with the DROP PACKAGE statement. We will discuss packages in the chapter 'PL/SQL - Packages'.

PL/SQL subprograms are named PL/SQL blocks that can be invoked with a set of parameters. PL/SQL provides two kinds of subprograms:

Functions: these subprograms return a single value, mainly used to compute and return a value.

Procedures: these subprograms do not return a value directly, mainly used to perform an action.

This chapter is going to cover important aspects of a PL/SQL procedure and we will cover PL/SQL function in next chapter.

Parts of a PL/SQL Subprogram Each PL/SQL subprogram has a name, and may have a parameter list. Like anonymous PL/SQL blocks and, the named blocks a subprograms will also have following three parts:

S.No	Parts & Description
1	**Declarative Part** It is an optional part. However, the declarative part for a sub-program does not start with the DECLARE keyword. It contains declarations o f types, cursors, constants, variables, exceptions, and nested subprograms. These items are local to the subprogram and cease to exist whe n the subprogram completes execution.
2	**Executable Part** This is a mandatory part and contains statements that perform the designated action.
3	**Exception-handling** This is again an optional part. It contains the code that handles run-time errors.

0.1.1 Creating a Procedure

A procedure is created with the CREATE OR REPLACE PROCEDURE statement. The simplified syntax for the CREATE OR REPLACE PROCEDURE statement is as follows:

```
CREATE [OR REPLACE] PROCEDURE procedure_name
[(parameter_name [IN | OUT | IN OUT] type [, ...])]
{IS | AS}
BEGIN
  < procedure_body >
END procedure_name;
```

Where,

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

Example: The following example creates a simple procedure that displays the string 'Hello World!' on the screen when executed.

```
CREATE OR REPLACE PROCEDURE greetings
AS
BEGIN
   dbms_output.put_line('Hello World!');
END;
/
```

When above code is executed using SQL prompt, it will produce the following result: *Output*: Procedure created.

Executing a Standalone Procedure

A standalone procedure can be called in two ways:

1. Using the EXECUTE keyword
2. Calling the name of the procedure from a PL/SQL block

The above procedure named 'greetings' can be called with the EXECUTE keyword as:

EXECUTE greetings;

The above call would display:

Hello World PL/SQL procedure successfully completed.

The procedure can also be called from another PL/SQL block:

```
BEGIN
   greetings;
END;
/
```

The above call would display:

Hello World PL/SQL procedure successfully completed.

Deleting a Standalone Procedure

A standalone procedure is deleted with the DROP PROCEDURE statement. Syntax for deleting a procedure is:

DROP PROCEDURE procedure-name; So you can drop greetings procedure by using the following statement:

```
BEGIN
   DROP PROCEDURE greetings;
END;
/
```

S.N.	Parameter Mode & Description
1	IN An IN parameter lets you pass a value to the subprogram. *It is a read-only parameter.* Inside the subprogram, an IN parameter acts like a constant. It cannot be assigned a value. You can pass a constant, literal, initialized variable, or expression as an IN parameter. You can also initialize it to a default value; however, in that case, it is omitted from the subprogram call. *It is the default mode of parameter passing. Parameters are passed by reference.*
2	OUT An OUT parameter returns a value to the calling program. Inside the subprogram, an OUT parameter acts like a variable. You can change its value and reference the value after assigning it. *The actual parameter must be variable and it is passed by value.*
3	IN OUT An IN OUT parameter passes an initial value to a subprogram and returns an updated value to the caller. It can be assigned a value and its value can be read. The actual parameter corresponding to an IN OUT formal parameter must be a variable, not a constant or an expression. Formal parameter must be assigned a value.*Actual parameter is passed by value.*

0.1.2 Parameter Modes in PL/SQL Subprograms

IN & OUT Mode Example 1

This program finds the minimum of two values, here procedure takes two numbers using IN mode and returns their minimum using OUT parameters.

```
DECLARE
   a number;
   b number;
   c number;

PROCEDURE findMin(x IN number, y IN number, z OUT number) IS
BEGIN
   IF x < y THEN
      z:= x;
   ELSE
      z:= y;
   END IF;
END;

BEGIN
```

```
   a:= 23;
   b:= 45;
   findMin(a, b, c);
   dbms_output.put_line(' Minimum of (23, 45) : ' || c);
END;
/
```

When the above code is executed at SQL prompt, it produces the following result: *Output*: Minimum of (23, 45) : 23 PL/SQL procedure successfully completed.

IN & OUT Mode Example 2

This procedure computes the square of value of a passed value. This example shows how we can use same parameter to accept a value and then return another result.

```
DECLARE
   a number;
PROCEDURE squareNum(x IN OUT number) IS
BEGIN
  x := x * x;
END;
BEGIN
   a:= 23;
   squareNum(a);
   dbms_output.put_line(' Square of (23): ' || a);
END;
/
```

When the above code is executed at SQL prompt, it produces the following result: *Output*: Square of (23): 529 PL/SQL procedure successfully completed.

0.1.3 Methods for Passing Parameters

Actual parameters could be passed in three ways:

1. Positional notation
2. Named notation
3. Mixed notation

POSITIONAL NOTATION

In positional notation, you can call the procedure as:

findMin(a, b, c, d);

In positional notation, the first actual parameter is substituted for the first formal parameter; the second actual parameter is substituted for the second formal parameter, and so on. So, a is substituted for x, b is substituted for y, c is substituted for z and d is substituted for m.

NAMED NOTATION

In named notation, the actual parameter is associated with the formal parameter using the arrow symbol (=>). So the procedure call would look like:

findMin(x=>a, y=>b, z=>c, m=>d);

MIXED NOTATION

In mixed notation, you can mix both notations in procedure call; however, the positional notation should precede the named notation.

The following call is legal:

findMin(a, b, c, m=>d);

But this is not legal:

findMin(x=>a, b, c, d);

Session 4 - DRL Assignments (Practice Session)