1.Comparitive study of various Database Management System

A comprehensive comparative study of various database management systems (DBMS) involves evaluating different systems based on various criteria such as data model, scalability, performance, security, ease of use, and cost. Below is a template for such a study, considering popular database management systems like MySQL, PostgreSQL, MongoDB, Microsoft SQL Server, and Oracle Database.

1. **Data Model:**
   - Explain the data models each DBMS follows.
   - Discuss how relational databases (MySQL, PostgreSQL, SQL Server, Oracle) organize data into tables, while MongoDB uses a document-oriented model with BSON documents.
2. **Open Source vs. Commercial:**
   - Differentiate between open-source (e.g., MySQL, PostgreSQL, MongoDB) and commercial (e.g., SQL Server, Oracle) database systems.
   - Discuss implications for licensing costs and community support.
3. **Programming Language Integration:**
   - Discuss how well each DBMS integrates with various programming languages.
   - Highlight that SQL-based systems have widespread compatibility, while MongoDB uses BSON and supports multiple languages.
4. **Scalability:**
   - Discuss scalability options for each DBMS.
   - Mention horizontal scaling (sharding, replication) and vertical scaling capabilities.
5. **Performance:**
   - Evaluate the performance characteristics of each DBMS.
   - Consider factors like read and write speed, query optimization, and overall system efficiency.
6. **Security:**
   - Examine the security features of each DBMS.
   - Discuss authentication, authorization, encryption, and any additional security measures.
7. **Ease of Use:**
   - Assess the user-friendliness of each DBMS.
   - Discuss community support, documentation, and the learning curve associated with each system.
8. **Community and Support:**
   - Compare the strength of the open-source communities for each DBMS.
   - Discuss the availability and quality of commercial support options.
9. **Use Cases:**
   - Identify the types of projects each DBMS is best suited for.
   - Consider transactional systems, analytics, real-time applications, etc.

10. **Cost:**
    - Break down the cost structures for each DBMS.
    - Consider licensing fees, support costs, and any additional expenses.
11. **Replication and High Availability:**
    - Evaluate the replication and high availability features of each DBMS.
    - Discuss how each system ensures data consistency and availability in distributed environments.
12. **Consistency and ACID Compliance:**
    - Discuss how each DBMS maintains consistency and adheres to ACID properties.
    - Consider the level of consistency and isolation provided in different systems.
13. **Data Query and Indexing:**
    - Assess the capabilities for querying and indexing in each DBMS.
    - Discuss the flexibility and efficiency of the query language and indexing mechanisms.
14. **Licensing and Vendor Lock-In:**
    - Discuss the licensing models of each DBMS.
    - Consider the potential for vendor lock-in and how it might impact long-term plans.
15. **Trends and Future Development:**
    - Provide insights into the current trends and ongoing development efforts for each DBMS.
    - Consider the responsiveness of each system to emerging technologies and industry needs.
16. **Conclusion:**
    - Summarize key findings.
    - Provide recommendations based on specific project requirements.
    - Emphasize the importance of considering factors beyond the feature set, such as community support and future development plans.

2.Data Definition Language(DDL), Data Manipulation Language (DML), And Data Control Language(DCL)

DDL : 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 the database. DDL is a set of SQL commands used to create, modify, and delete database structures but not data. These commands are normally not used by a general user, who should be accessing the database via an application.

- List of DDL commands:

- **CREATE**: This command is used to create the database or its objects (like table, index, function, views, store procedure, and triggers).
- **DROP**: This command is used to delete objects from the database.
- **ALTER:** This is used to alter the structure of the database.
- **TRUNCATE:** This is used to remove all records from a table, including all spaces allocated for the records are removed.
- **COMMENT**: This is used to add comments to the data dictionary.
- **RENAME:** This is used to rename an object existing in the database

DML: The SQL commands that deal with the manipulation of data present in the database belong to DML or Data Manipulation Language and this includes most of the SQL statements. It is the component of the SQL statement that controls access to data and to the database. Basically, DCL statements are grouped with DML statements.

- List of DML commands:

- **INSERT**: It is used to insert data into a table.
- **UPDATE:** It is used to update existing data within a table.
- **DELETE**: It is used to delete records from a database table.
- **LOCK:** Table control concurrency.
- **CALL:** Call a PL/SQL or JAVA subprogram.
- **EXPLAIN PLAN:** It describes the access path to data.

DCL: DCL includes commands such as GRANT and REVOKE which mainly deal with the rights, permissions, and other controls of the database system.

- List of DCL commands:

- **GRANT:** This command gives users access privileges to the database.
- **Syntax:**
  GRANT SELECT, UPDATE ON MY_TABLE TO SOME_USER, ANOTHER_USER;

- **REVOKE:** This command withdraws the user's access privileges given by using the GRANT command.
- **Syntax:**

  REVOKE SELECT, UPDATE ON MY_TABLE FROM USER1, USER2;

3.how to apply Constraints at various levels.

SQL constraints are used to specify rules for data in a table.

SQL CONSTRAINTS:

Constraints can be specified when the table is created with the CREATE TABLE statement, or after the table is created with the ALTER TABLE statement.

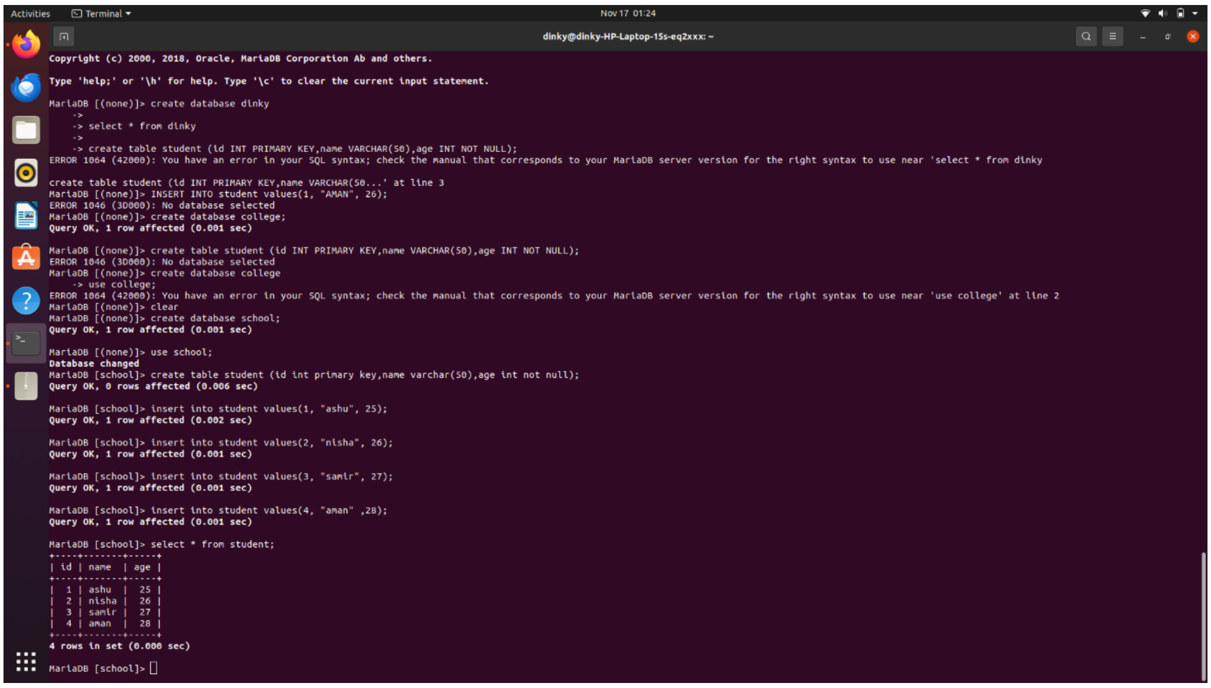
The following constraints are commonly used in SQL:

- NOT NULL - Ensures that a column cannot have a NULL value
- UNIQUE - Ensures that all values in a column are different
- PRIMARY KEY - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table
- FOREIGN KEY - Prevents actions that would destroy links between tables
- CHECK - Ensures that the values in a column satisfies a specific condition
- DEFAULT - Sets a default value for a column if no value is specified
- CREATE INDEX - Used to create and retrieve data from the database very quickly

PRIMARY KEY: The `PRIMARY KEY` constraint uniquely identifies each record in a table.

- Primary keys must contain UNIQUE values, and cannot contain NULL values.

- A table can have only ONE primary key; and in the table, this primary key can consist of single or multiple columns (fields).



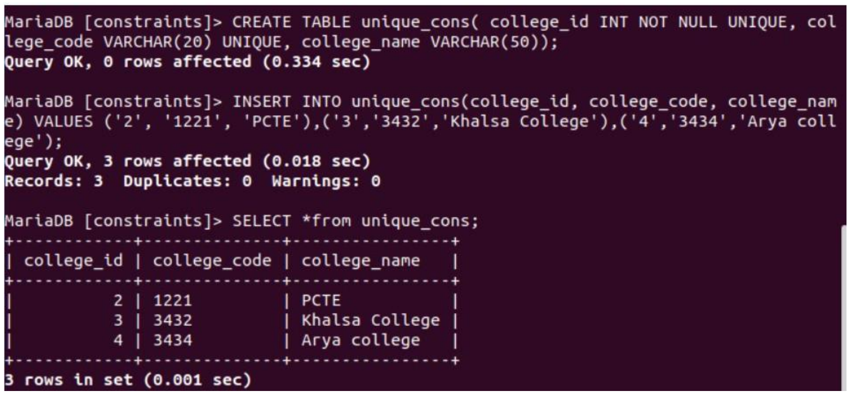
```
Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> create database dinky
    ->
    -> select * from dinky
    ->
    -> create table student (id INT PRIMARY KEY,name VARCHAR(50),age INT NOT NULL);
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server version for the right syntax to use near 'select * from dinky

create table student (id INT PRIMARY KEY,name VARCHAR(50...' at line 3
MariaDB [(none)]> INSERT INTO student values(1, "AMAN", 26);
ERROR 1046 (3D000): No database selected
MariaDB [(none)]> create database college;
Query OK, 1 row affected (0.001 sec)

MariaDB [(none)]> create table student (id INT PRIMARY KEY,name VARCHAR(50),age INT NOT NULL);
ERROR 1046 (3D000): No database selected
MariaDB [(none)]> create database college
    -> use college;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server version for the right syntax to use near 'use college' at line 2
MariaDB [(none)]> clear
MariaDB [(none)]> create database school;
Query OK, 1 row affected (0.001 sec)

MariaDB [(none)]> use school;
Database changed
MariaDB [school]> create table student (id int primary key,name varchar(50),age int not null);
Query OK, 0 rows affected (0.006 sec)

MariaDB [school]> insert into student values(1, "ashu", 25);
Query OK, 1 row affected (0.002 sec)

MariaDB [school]> insert into student values(2, "nisha", 26);
Query OK, 1 row affected (0.001 sec)

MariaDB [school]> insert into student values(3, "samir", 27);
Query OK, 1 row affected (0.001 sec)

MariaDB [school]> insert into student values(4, "aman" ,28);
Query OK, 1 row affected (0.001 sec)

MariaDB [school]> select * from student;
+----+-------+-----+
| id | name  | age |
+----+-------+-----+
|  1 | ashu  |  25 |
|  2 | nisha |  26 |
|  3 | samir |  27 |
|  4 | aman  |  28 |
+----+-------+-----+
4 rows in set (0.000 sec)

MariaDB [school]> 
```

UNIQUE CONSTRAINTS: The `UNIQUE` constraint ensures that all values in a column are different.

- Both the UNIQUE and PRIMARY KEY constraints provide a guarantee for uniqueness for a column or set of columns.

- A PRIMARY KEY constraint automatically has a UNIQUE constraint.

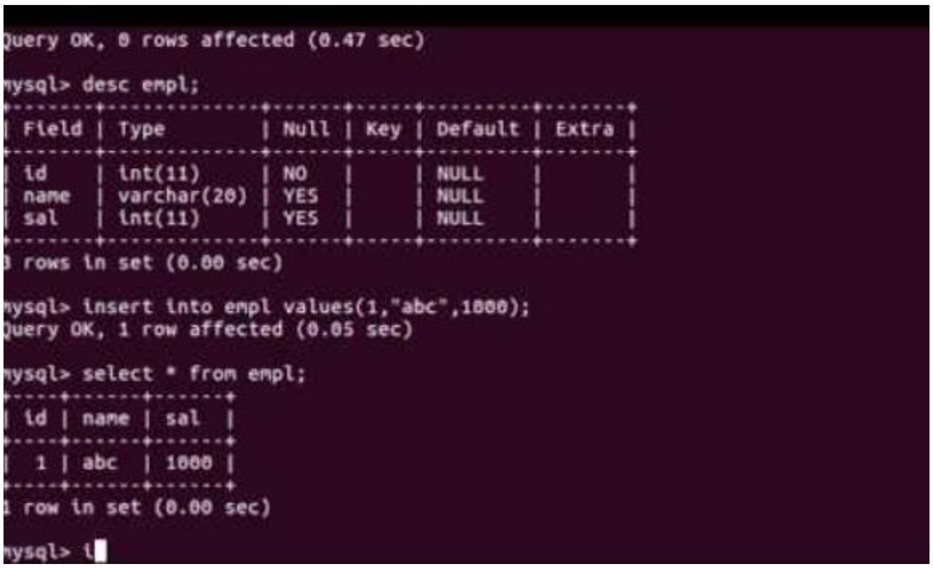
- However, you can have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table.



```
MariaDB [constraints]> CREATE TABLE unique_cons( college_id INT NOT NULL UNIQUE, col
lege_code VARCHAR(20) UNIQUE, college_name VARCHAR(50));
Query OK, 0 rows affected (0.334 sec)

MariaDB [constraints]> INSERT INTO unique_cons(college_id, college_code, college_nam
e) VALUES ('2', '1221', 'PCTE'),('3','3432','Khalsa College'),('4','3434','Arya coll
ege');
Query OK, 3 rows affected (0.018 sec)
Records: 3  Duplicates: 0  Warnings: 0

MariaDB [constraints]> SELECT *from unique_cons;
+------------+--------------+----------------+
| college_id | college_code | college_name   |
+------------+--------------+----------------+
|          2 | 1221         | PCTE           |
|          3 | 3432         | Khalsa College |
|          4 | 3434         | Arya college   |
+------------+--------------+----------------+
3 rows in set (0.001 sec)
```

NOT NULL : The NOT NULL constraint enforces a column to NOT accept NULL values.

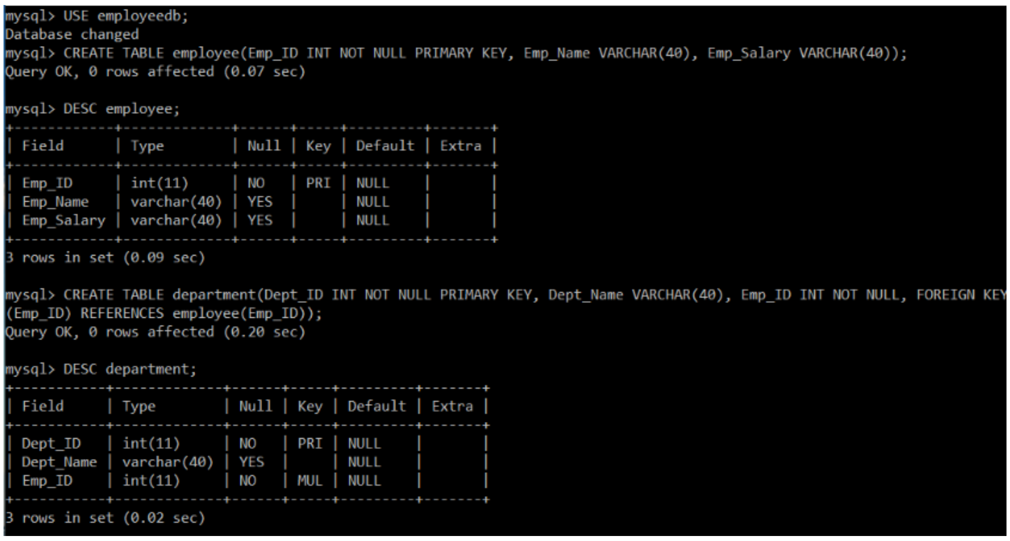
This enforces a field to always contain a value, which means that you cannot insert a new record, or update a record without adding a value to this field.



```
Query OK, 0 rows affected (0.47 sec)

ysql> desc empl;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| id    | int(11)     | NO   |     | NULL    |       |
| name  | varchar(20) | YES  |     | NULL    |       |
| sal   | int(11)     | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)

ysql> insert into empl values(1,"abc",1000);
Query OK, 1 row affected (0.05 sec)

ysql> select * from empl;
+----+------+------+
| id | name | sal  |
+----+------+------+
|  1 | abc  | 1000 |
+----+------+------+
1 row in set (0.00 sec)

ysql> i
```

FOREIGN KEY: The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables.

A FOREIGN KEY is a field (or collection of fields) in one table, that refers to the PRIMARY KEY in another table.The table with the foreign key is called the child table, and the table with the primary key is called the referenced or parent table.



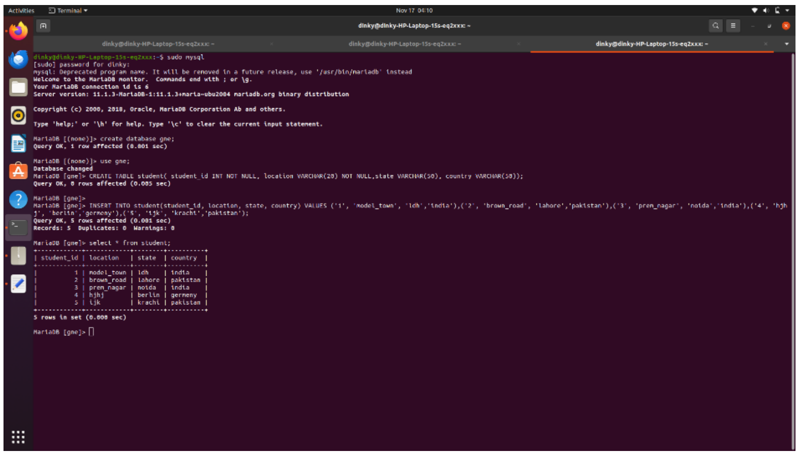
```
mysql> USE employeedb;
Database changed
mysql> CREATE TABLE employee(Emp_ID INT NOT NULL PRIMARY KEY, Emp_Name VARCHAR(40), Emp_Salary VARCHAR(40));
Query OK, 0 rows affected (0.07 sec)

mysql> DESC employee;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| Emp_ID     | int(11)     | NO   | PRI | NULL    |       |
| Emp_Name   | varchar(40) | YES  |     | NULL    |       |
| Emp_Salary | varchar(40) | YES  |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+
3 rows in set (0.09 sec)

mysql> CREATE TABLE department(Dept_ID INT NOT NULL PRIMARY KEY, Dept_Name VARCHAR(40), Emp_ID INT NOT NULL, FOREIGN KEY
(Emp_ID) REFERENCES employee(Emp_ID));
Query OK, 0 rows affected (0.20 sec)

mysql> DESC department;
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| Dept_ID   | int(11)     | NO   | PRI | NULL    |       |
| Dept_Name | varchar(40) | YES  |     | NULL    |       |
| Emp_ID    | int(11)     | NO   | MUL | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
3 rows in set (0.02 sec)
```

- CHECK: The `CHECK` constraint is used to limit the value range that can be placed in a column.
- If you define a `CHECK` constraint on a column it will allow only certain values for this column.
- If you define a `CHECK` constraint on a table it can limit the values in certain columns based on values in other columns in the row.

4.View data in the required form using Operators, Functions, Joins: **AND Operator (&& and AND):**

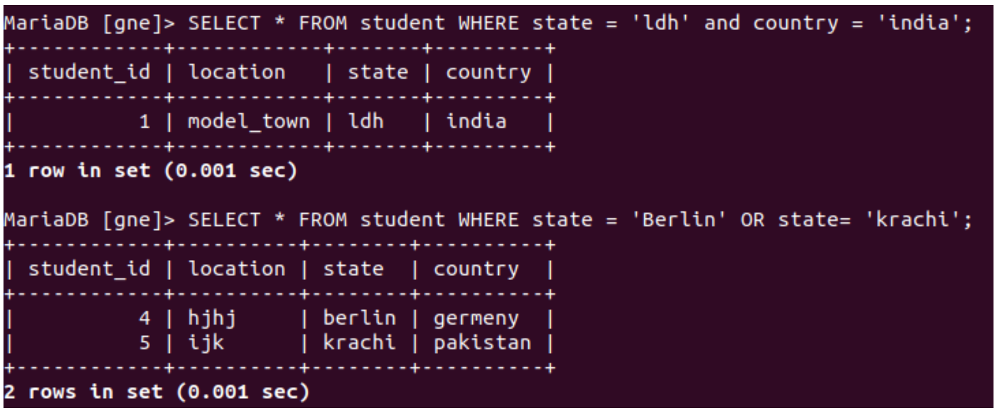
•   The AND operator is used to combine multiple conditions, and all conditions must be true for the overall condition to be true.

•   The && operator can also be used as a shorthand for AND.



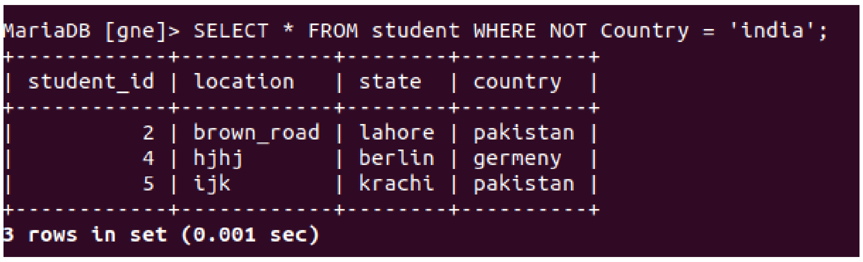
```
dinky@dinky-HP-Laptop-15s-eq2xxx:~$ sudo mysql
[sudo] password for dinky:
mysql: Deprecated program name. It will be removed in a future release, use '/usr/bin/mariadb' instead
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 6
Server version: 11.1.3-MariaDB-1:11.1.3+maria~ubu2004 mariadb.org binary distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> create database gne;
Query OK, 1 row affected (0.001 sec)

MariaDB [(none)]> use gne;
Database changed
MariaDB [gne]> CREATE TABLE student( student_id INT NOT NULL, location VARCHAR(20) NOT NULL,state VARCHAR(50), country VARCHAR(50));
Query OK, 0 rows affected (0.005 sec)

MariaDB [gne]>
MariaDB [gne]> INSERT INTO student(student_id, location, state, country) VALUES ('1', 'model_town', 'ldh','india'),('2', 'brown_road', 'lahore','pakistan'),('3', 'prem_nagar', 'noida','india'),('4', 'hjh
j', 'berlin','germeny'),('5', 'ijk', 'krachi','pakistan');
Query OK, 5 rows affected (0.001 sec)
Records: 5  Duplicates: 0  Warnings: 0

MariaDB [gne]> select * from student;
+------------+------------+--------+----------+
| student_id | location   | state  | country  |
+------------+------------+--------+----------+
|          1 | model_town | ldh    | india    |
|          2 | brown_road | lahore | pakistan |
|          3 | prem_nagar | noida  | india    |
|          4 | hjhj       | berlin | germeny  |
|          5 | ijk        | krachi | pakistan |
+------------+------------+--------+----------+
5 rows in set (0.000 sec)

MariaDB [gne]> 
```

- **OR OPERATOR:** The OR operator is used to combine multiple conditions, and at least one condition must be true for the overall condition to be **true.**
- The || operator can also be used as a shorthand for OR.



```
MariaDB [gne]> SELECT * FROM student WHERE state = 'ldh' and country = 'india';
+------------+------------+-------+---------+
| student_id | location   | state | country |
+------------+------------+-------+---------+
|          1 | model_town | ldh   | india   |
+------------+------------+-------+---------+
1 row in set (0.001 sec)

MariaDB [gne]> SELECT * FROM student WHERE state = 'Berlin' OR state= 'krachi';
+------------+----------+--------+----------+
| student_id | location | state  | country  |
+------------+----------+--------+----------+
|          4 | hjhj     | berlin | germeny  |
|          5 | ijk      | krachi | pakistan |
+------------+----------+--------+----------+
2 rows in set (0.001 sec)
```

NOT OPERATOR: The NOT operator is used to negate a condition, returning the opposite of the specified condition



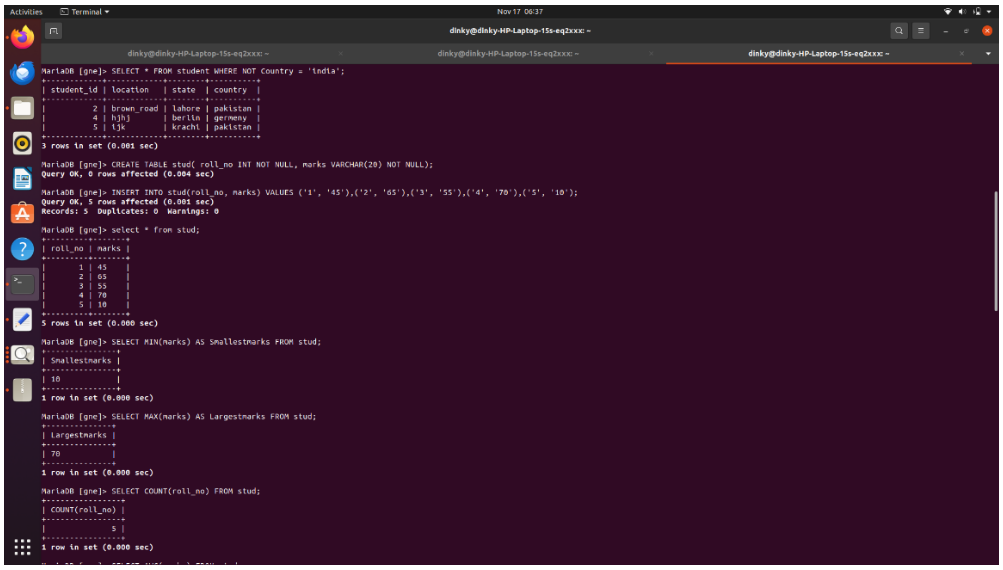
```
MariaDB [gne]> SELECT * FROM student WHERE NOT Country = 'india';
+------------+------------+--------+----------+
| student_id | location   | state  | country  |
+------------+------------+--------+----------+
|          2 | brown_road | lahore | pakistan |
|          4 | hjhj       | berlin | germeny  |
|          5 | ijk        | krachi | pakistan |
+------------+------------+--------+----------+
3 rows in set (0.001 sec)
```

- **FUNCTIONS:**

  MySQL MIN(),MAX() Functions:

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

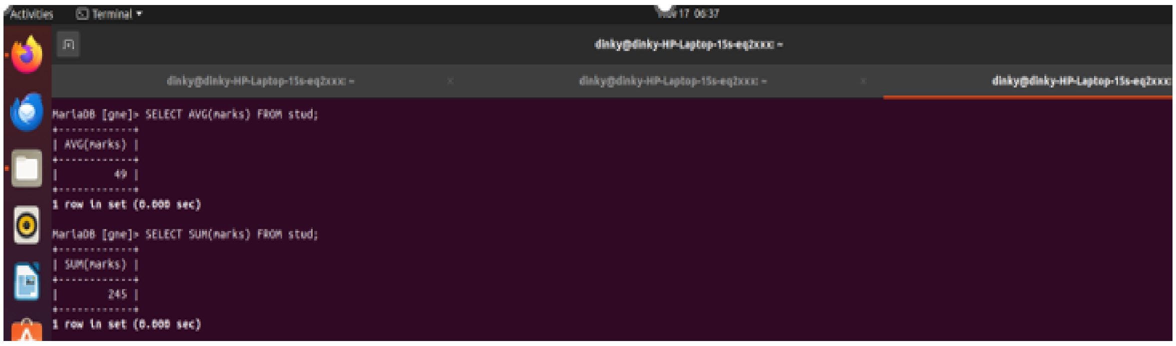
- The `MAX()` function returns the largest value of the selected column.



```
MariaDB [gne]> SELECT * FROM student WHERE NOT Country = 'india';
+------------+------------+--------+----------+
| student_id | location   | state  | country  |
+------------+------------+--------+----------+
|          2 | brown_road | lahore | pakistan |
|          4 | hjhj       | berlin | germeny  |
|          5 | ijk        | krachi | pakistan |
+------------+------------+--------+----------+
3 rows in set (0.001 sec)

MariaDB [gne]> CREATE TABLE stud( roll_no INT NOT NULL, marks VARCHAR(20) NOT NULL);
Query OK, 0 rows affected (0.004 sec)

MariaDB [gne]> INSERT INTO stud(roll_no, marks) VALUES ('1', '45'),('2', '65'),('3', '55'),('4', '70'),('5', '10');
Query OK, 5 rows affected (0.001 sec)
Records: 5  Duplicates: 0  Warnings: 0

MariaDB [gne]> select * from stud;
+---------+-------+
| roll_no | marks |
+---------+-------+
|       1 | 45    |
|       2 | 65    |
|       3 | 55    |
|       4 | 70    |
|       5 | 10    |
+---------+-------+
5 rows in set (0.000 sec)

MariaDB [gne]> SELECT MIN(marks) AS Smallestmarks FROM stud;
+---------------+
| Smallestmarks |
+---------------+
| 10            |
+---------------+
1 row in set (0.000 sec)

MariaDB [gne]> SELECT MAX(marks) AS Largestmarks FROM stud;
+--------------+
| Largestmarks |
+--------------+
| 70           |
+--------------+
1 row in set (0.000 sec)

MariaDB [gne]> SELECT COUNT(roll_no) FROM stud;
+----------------+
| COUNT(roll_no) |
+----------------+
|              5 |
+----------------+
1 row in set (0.000 sec)
```

MySQL COUNT(), AVG() and SUM() Functions:

- The `COUNT()` function returns the number of rows that matches a specified criterion.
- The `AVG()` function returns the average value of a numeric column
- The `SUM()` function returns the total sum of a numeric column



```
MariaDB [gne]> SELECT AVG(marks) FROM stud;
+------------+
| AVG(marks) |
+------------+
|         49 |
+------------+
1 row in set (0.000 sec)

MariaDB [gne]> SELECT SUM(marks) FROM stud;
+------------+
| SUM(marks) |
+------------+
|        245 |
+------------+
1 row in set (0.000 sec)
```

JOINS:

INNER JOIN:

The `INNER JOIN` keyword selects records that have matching values in both tables.

LEFT JOIN:

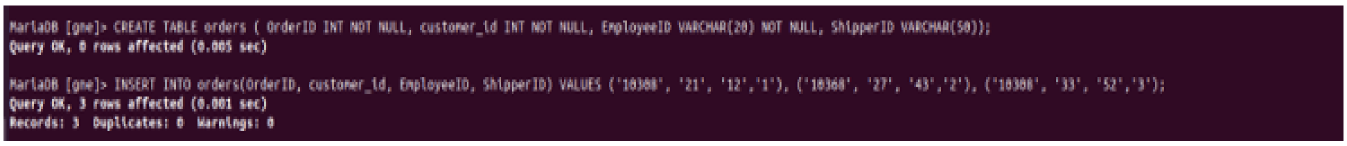
Returns all records from the left table, and the matched records from the right table

RIGHT JOIN:

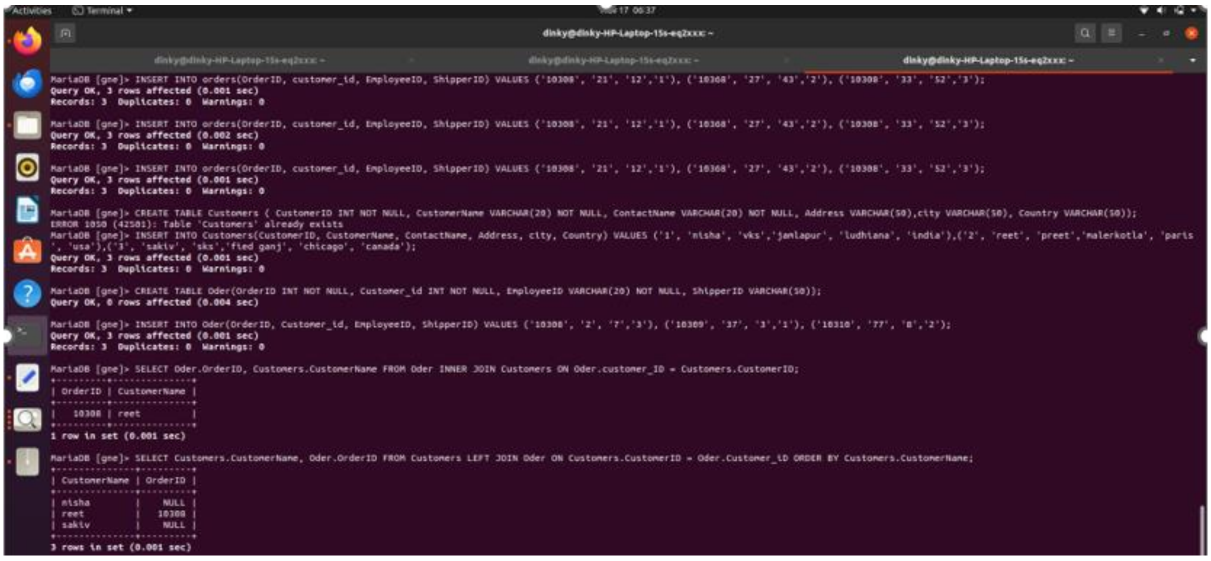
Returns all records from the right table, and the matched records from the left table

CROSS JOIN:

Returns all records from both tables



```
MariaDB [gne]> CREATE TABLE orders ( OrderID INT NOT NULL, customer_id INT NOT NULL, EmployeeID VARCHAR(20) NOT NULL, ShipperID VARCHAR(50));
Query OK, 0 rows affected (0.005 sec)

MariaDB [gne]> INSERT INTO orders(OrderID, customer_id, EmployeeID, ShipperID) VALUES ('10308', '21', '12','1'), ('10368', '27', '43','2'), ('10308', '33', '52','3');
Query OK, 3 rows affected (0.001 sec)
Records: 3  Duplicates: 0  Warnings: 0
```



```
MariaDB [gne]> INSERT INTO orders(OrderID, customer_id, EmployeeID, ShipperID) VALUES ('10308', '21', '12','1'), ('10368', '27', '43','2'), ('10308', '33', '52','3');
Query OK, 3 rows affected (0.001 sec)
Records: 3  Duplicates: 0  Warnings: 0

MariaDB [gne]> INSERT INTO orders(OrderID, customer_id, EmployeeID, ShipperID) VALUES ('10308', '21', '12','1'), ('10368', '27', '43','2'), ('10308', '33', '52','3');
Query OK, 3 rows affected (0.002 sec)
Records: 3  Duplicates: 0  Warnings: 0

MariaDB [gne]> INSERT INTO orders(OrderID, customer_id, EmployeeID, ShipperID) VALUES ('10308', '21', '12','1'), ('10368', '27', '43','2'), ('10308', '33', '52','3');
Query OK, 3 rows affected (0.001 sec)
Records: 3  Duplicates: 0  Warnings: 0

MariaDB [gne]> CREATE TABLE Customers ( CustomerID INT NOT NULL, CustomerName VARCHAR(20) NOT NULL, ContactName VARCHAR(20) NOT NULL, Address VARCHAR(50),city VARCHAR(50), Country VARCHAR(50));
ERROR 1050 (42S01): Table 'Customers' already exists
MariaDB [gne]> INSERT INTO Customers(CustomerID, CustomerName, ContactName, Address, city, Country) VALUES ('1', 'nisha', 'vks','jamlapur', 'ludhiana', 'india'),('2', 'reet', 'preet','malerkotla', 'paris
', 'usa'),('3', 'sakiv', 'sks','fied ganj', 'chicago', 'canada');
Query OK, 3 rows affected (0.001 sec)
Records: 3  Duplicates: 0  Warnings: 0

MariaDB [gne]> CREATE TABLE Oder(OrderID INT NOT NULL, Customer_id INT NOT NULL, EmployeeID VARCHAR(20) NOT NULL, ShipperID VARCHAR(50));
Query OK, 0 rows affected (0.004 sec)

MariaDB [gne]> INSERT INTO Oder(OrderID, Customer_id, EmployeeID, ShipperID) VALUES ('10308', '2', '7','3'), ('10309', '37', '3','1'), ('10310', '77', '8','2');
Query OK, 3 rows affected (0.001 sec)
Records: 3  Duplicates: 0  Warnings: 0

MariaDB [gne]> SELECT Oder.OrderID, Customers.CustomerName FROM Oder INNER JOIN Customers ON Oder.customer_ID = Customers.CustomerID;
+---------+--------------+
| OrderID | CustomerName |
+---------+--------------+
|   10308 | reet         |
+---------+--------------+
1 row in set (0.001 sec)

MariaDB [gne]> SELECT Customers.CustomerName, Oder.OrderID FROM Customers LEFT JOIN Oder ON Customers.CustomerID = Oder.Customer_ID ORDER BY Customers.CustomerName;
+--------------+---------+
| CustomerName | OrderID |
+--------------+---------+
| nisha        |    NULL |
| reet         |   10308 |
| sakiv        |    NULL |
+--------------+---------+
3 rows in set (0.001 sec)
```

5.**How to apply conditional controls in PL/SQL:**

4.1 Conditional Selection Statements

The conditional selection statements, IF and CASE, run different statements for different data values.

The IF statement either runs or skips a sequence of one or more statements, depending on a condition. The IF statement has these forms:

- IF THEN
- IF THEN ELSE
- IF THEN ELSIF

The CASE statement chooses from a sequence of conditions, and runs the corresponding statement. The CASE statement has these forms:

Simple, which evaluates a single expression and compares it to several potential values.

IF THEN Statement:

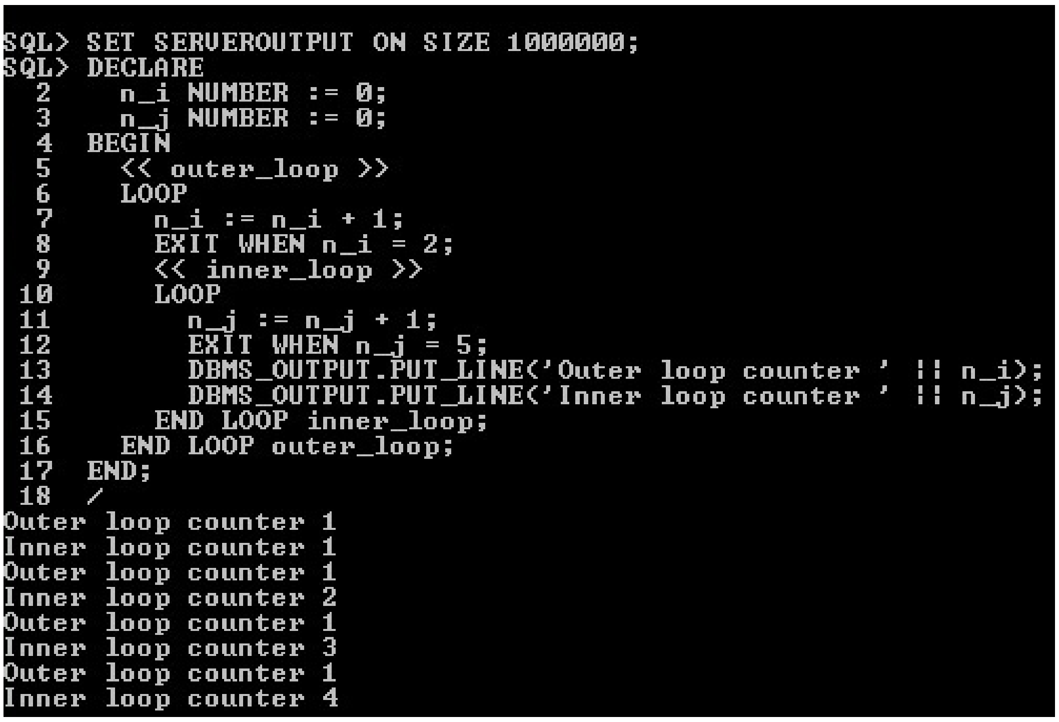
The IF THEN statement either runs or skips a sequence of one or more statements, depending on a condition.

IF THEN ELSE Statement:

If the value of *condition* is true, the *statements* run; otherwise, the *else_statements* run.

IF THEN ELSE Statement:

When an If ... Then ... Else statement is encountered, condition is tested. If condition is True , the statements following Then are executed. If condition is False , each ElseIf statement (if there are any) is evaluated in order.

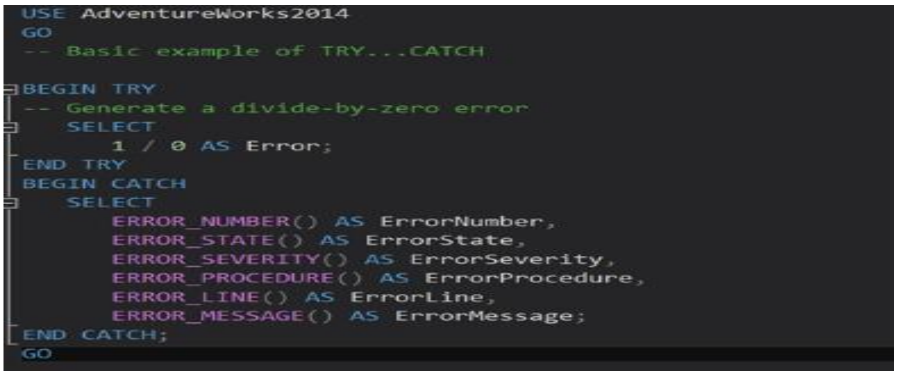


```
SQL> SET SERVEROUTPUT ON SIZE 1000000;
SQL> DECLARE
       n_i NUMBER := 0;
       n_j NUMBER := 0;
     BEGIN
       << outer_loop >>
       LOOP
         n_i := n_i + 1;
         EXIT WHEN n_i = 2;
         << inner_loop >>
         LOOP
           n_j := n_j + 1;
           EXIT WHEN n_j = 5;
           DBMS_OUTPUT.PUT_LINE('Outer loop counter ' || n_i);
           DBMS_OUTPUT.PUT_LINE('Inner loop counter ' || n_j);
         END LOOP inner_loop;
       END LOOP outer_loop;
     END;
     /
Outer loop counter 1
Inner loop counter 1
Outer loop counter 1
Inner loop counter 2
Outer loop counter 1
Inner loop counter 3
Outer loop counter 1
Inner loop counter 4
```

7. Error Handling using Internal Exceptions and External Exceptions:-

Error handling:-

Error handling in SQL Server gives us control over the Transact-SQL code. For example, when things go wrong, we get a chance to do something about it and possibly make it right again. SQL Server error handling can be as simple as just logging that something happened, or it could be us trying to fix an error. It can even be translating the error in SQL language because we all know how technical SQL Server error messages could get making no sense and hard to understand. Luckily, we have a chance to translate those messages into something more meaningful to pass on to the users, developers, etc.



```
USE AdventureWorks2014
GO
-- Basic example of TRY...CATCH

BEGIN TRY
-- Generate a divide-by-zero error
    SELECT
        1 / 0 AS Error;
END TRY
BEGIN CATCH
    SELECT
        ERROR_NUMBER() AS ErrorNumber,
        ERROR_STATE() AS ErrorState,
        ERROR_SEVERITY() AS ErrorSeverity,
        ERROR_PROCEDURE() AS ErrorProcedure,
        ERROR_LINE() AS ErrorLine,
        ERROR_MESSAGE() AS ErrorMessage;
END CATCH;
GO
```

Internal exception:- This exception is thrown when a SCALAR function does not return any row (value). Column name does not exist. A column that does not exist in a table is referenced in a query. Table already exists.

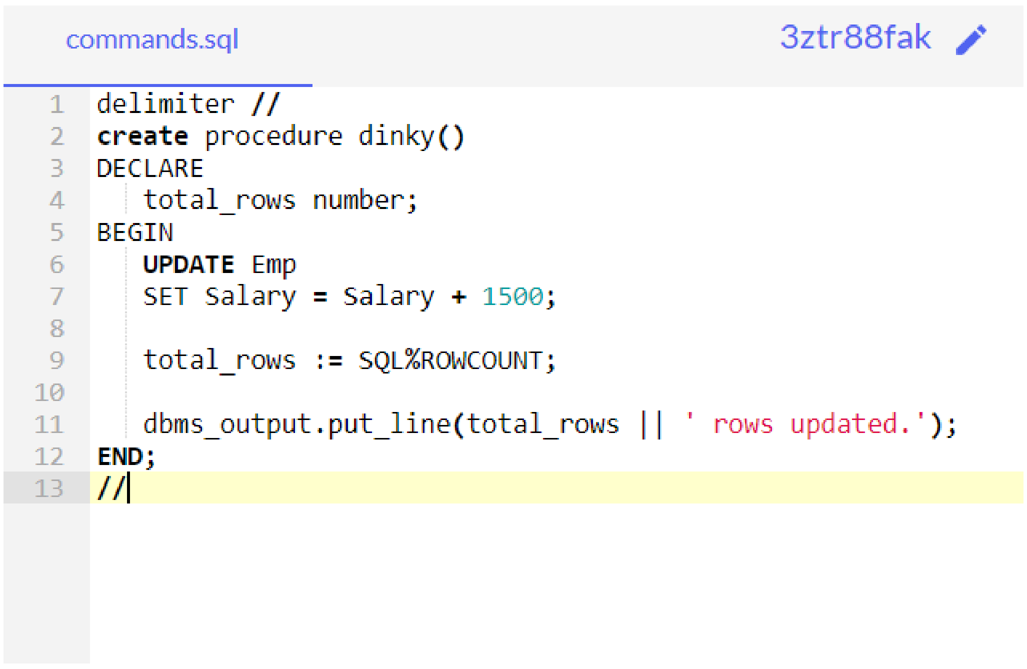
External exception:- The problem here is that the exception is generated outside the exception handeler. The parameters sent to the function is verified before the exception handler is defined. To catch that exception you need an exception handler around the call to the function.

8. Using various types of Cursors:-

Cursor is a Temporary Memory or Temporary Work Station. It is Allocated by Database Server at the Time of Performing DML(Data Manipulation Language) operations on the Table by the User. Cursors are used to store Database Tables.
There are 2 types of Cursors: Implicit Cursors, and Explicit Cursors. These are explained as following below.

1. **Implicit Cursors:** Implicit Cursors are also known as Default Cursors of SQL SERVER. These Cursors are allocated by SQL SERVER when the user performs DML operations.
2. **Explicit Cursors:** Explicit Cursors are Created by Users whenever the user requires them. Explicit Cursors are used for Fetching data from Table in Row-By-Row Manner.

Program of the courser:-



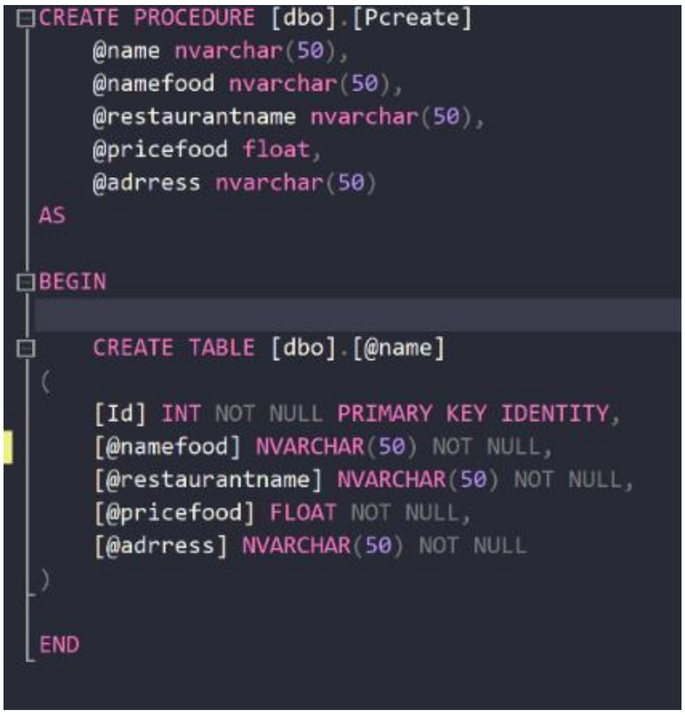
```
delimiter //
create procedure dinky()
DECLARE
    total_rows number;
BEGIN
    UPDATE Emp
    SET Salary = Salary + 1500;

    total_rows := SQL%ROWCOUNT;

    dbms_output.put_line(total_rows || ' rows updated.');
END;
//
```

9. How to run Stored Procedures and Functions

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

The following are the features of stored procedure in SQL Server:

- **Reduced Traffic:** A stored procedure reduces network traffic between the application and the database server, resulting in increased performance. It is because instead of sending several SQL statements, the application only needs to send the name of the stored procedure and its parameters.
- **Stronger Security:** The procedure is always secure because it manages which processes and activities we can perform. It removes the need for permissions to be granted at the database object level and simplifies the security layers.
- **Reusable:** Stored procedures are reusable. It reduces code inconsistency, prevents unnecessary rewrites of the same code, and makes the code transparent to all applications or users.
- **Easy Maintenance:** The procedures are easier to maintain without restarting or deploying the application.



```sql
CREATE PROCEDURE [dbo].[Pcreate]
    @name nvarchar(50),
    @namefood nvarchar(50),
    @restaurantname nvarchar(50),
    @pricefood float,
    @adrress nvarchar(50)
AS

BEGIN

    CREATE TABLE [dbo].[@name]
(
    [Id] INT NOT NULL PRIMARY KEY IDENTITY,
    [@namefood] NVARCHAR(50) NOT NULL,
    [@restaurantname] NVARCHAR(50) NOT NULL,
    [@pricefood] FLOAT NOT NULL,
    [@adrress] NVARCHAR(50) NOT NULL
)

END
```

Functions:-

In MySQL, Function can also be created. A function always returns a value using the return statement. The function can be used in SQL queries.

Syntax

```
1. CREATE FUNCTION function_name [ (parameter datatype [, parameter datatype]) ]
2. RETURNS return_datatype
3. BEGIN
4. Declaration_section
5. Executable_section
6. END;
```

Parameter:

Function_name: name of the function

Parameter: number of parameter. It can be one or more than one.

return_datatype: return value datatype of the function

declaration_section: all variables are declared.

executable_section: code for the function is written here.

10. Creating Packages and applying Triggers:-

A trigger is a stored procedure in a database that automatically invokes whenever a special event in the database occurs. For example, a trigger can be invoked when a row is inserted into a specified table or when specific table columns are updated in simple words a trigger is a collection of SQL statements with particular names that are stored in system memory. It belongs to a specific class of stored procedures that are automatically invoked in response to database server events. Every trigger has a table attached to it.

The following are the key differences between triggers and stored procedures:

1. Triggers cannot be manually invoked or executed.
2. There is no chance that triggers will receive parameters.
3. A transaction cannot be committed or rolled back inside a trigger.

Syntax:-
create trigger [trigger_name]

[before | after]

{insert | update | delete}

on [table_name]

[for each row]

[trigger_body]

11. Creating Arrays and Nested Tables:-
Nested tables are very similar to the PL/SQL tables, which are known in Oracle as index-by tables. Nested tables extend the functionality of index-by table by adding extra collection methods (known as table attributes for index-by tables) and by adding the ability to store nested tables within a database table, which is why they are called nested tables.

Nested tables can also be manipulated directly using SQL, and have additional predefined exceptions available.

Other than these extra features, the basic functionality of a nested table is the same as a PL/SQL table. A nested table can be thought as off as a database table with two columns-key and value. Like index-tables, nested tables can be sparse, and the keys do not have to be sequential.

Declaring a Nested Table

The syntax for creating a nested table type

```
TYPE table_name is TABLE OF table_type [NOT NULL];
```

where table_nameis the name of the new type, and table_type is the type of each element in the nested table. Table_type can be a built-in type, a user-defined object type, or an expression using % TYPE.

Arrays :-

PL/SQL provides a data structure called VARRAY, that can store fixed-size sequential collection of elements of the same type. It is used for storing an ordered collection of data. VARRAYS consists of contiguous memory location. That means the lowest address corresponds to the first element and highest address corresponds to the last element. Array is said to be a part of collection type of data and it stand for variable-size arrays. And then each element in a VARRAY has an index associated with it. It then has maximum size which can be changed dynamically.

Syntax:

```
CREATE OR REPLACE TYPE varray_type_name IS VARRAY(n) of
<element_type>
```