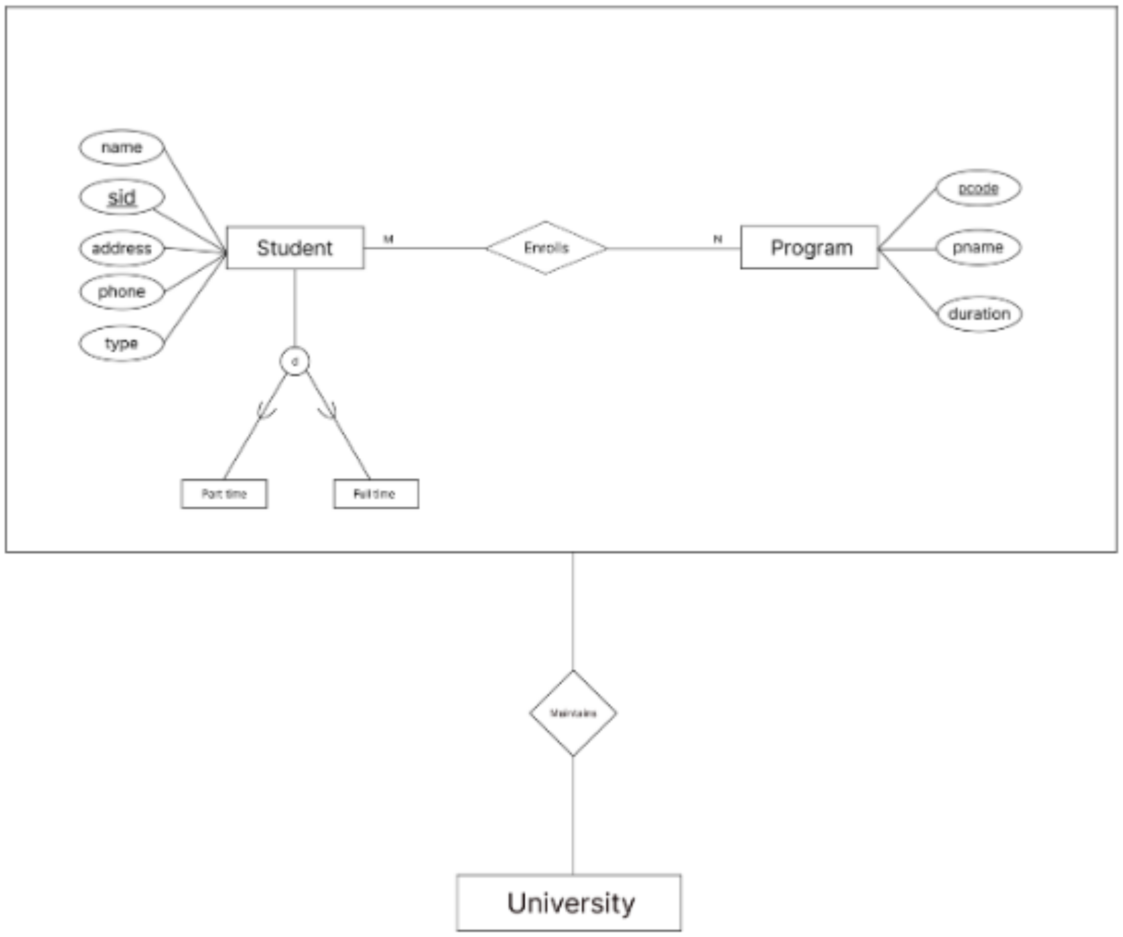
1. Develop an EER model for following

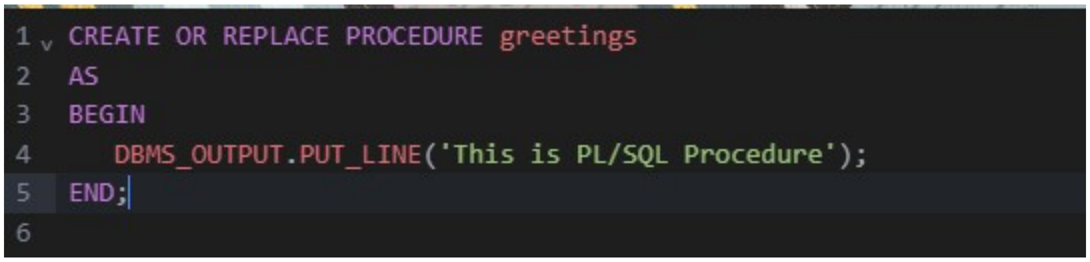
A university maintains records of its students and programs in which they have enrolled. It stores student id, name, address, and phone number of student and programs code, program name and dura;on of a program. A student is either a full-;me or a part-;me student (only one of the types). A student can register for many program and programs can have many students.



2. Apply PL/SQL for processing database. Give examples of Procedure, Functions and Triggers

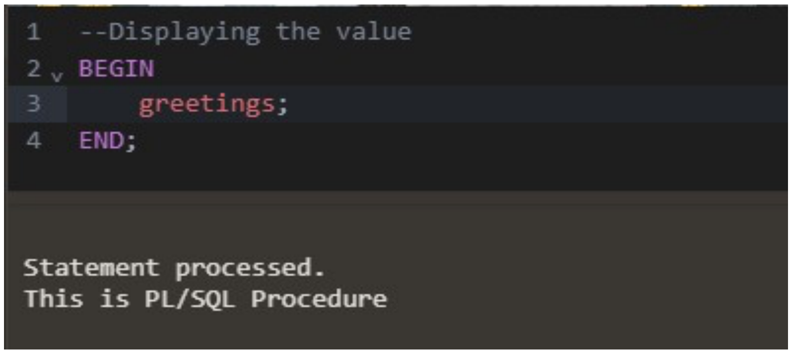
PL/SQL (Procedural Language/Structured Query Language) is a powerful extension of SQL used for writing procedural code in Oracle databases. You can use PL/SQL to create stored procedures, functions, and triggers for processing data in the database. Here are examples of each:

Procedure



```
CREATE OR REPLACE PROCEDURE greetings
AS
BEGIN
   DBMS_OUTPUT.PUT_LINE('This is PL/SQL Procedure');
END;
```

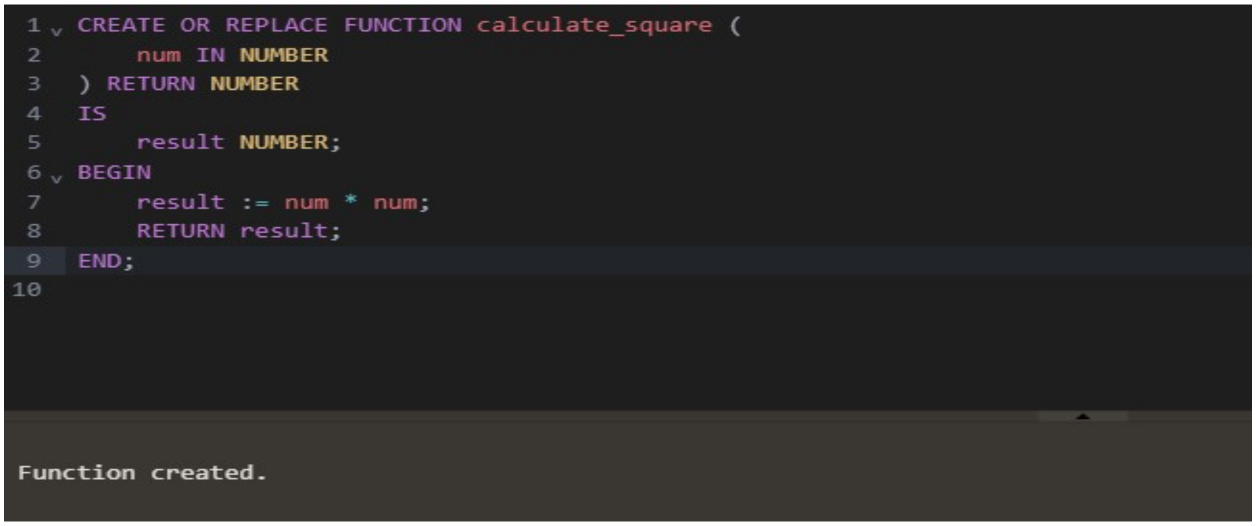
Output:



```
--Displaying the value
BEGIN
    greetings;
END;
```

```
Statement processed.
This is PL/SQL Procedure
```

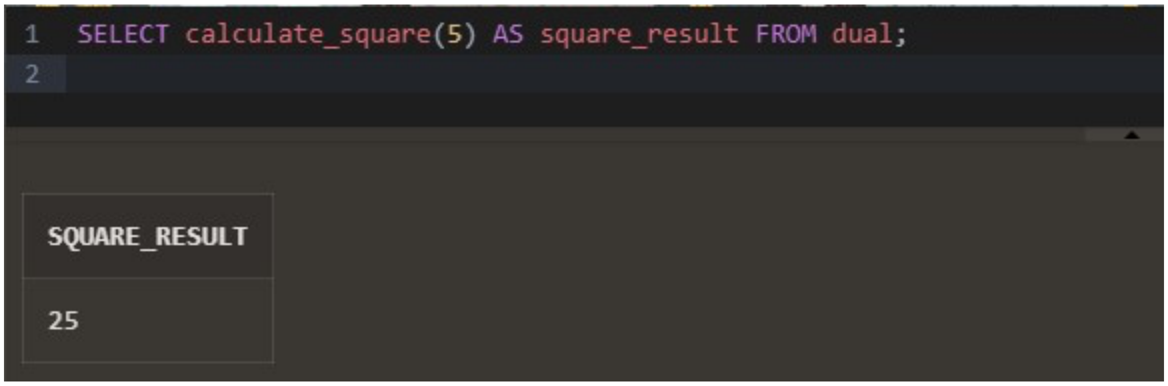
Functions



```
CREATE OR REPLACE FUNCTION calculate_square (
    num IN NUMBER
) RETURN NUMBER
IS
    result NUMBER;
BEGIN
    result := num * num;
    RETURN result;
END;
```

```
Function created.
```

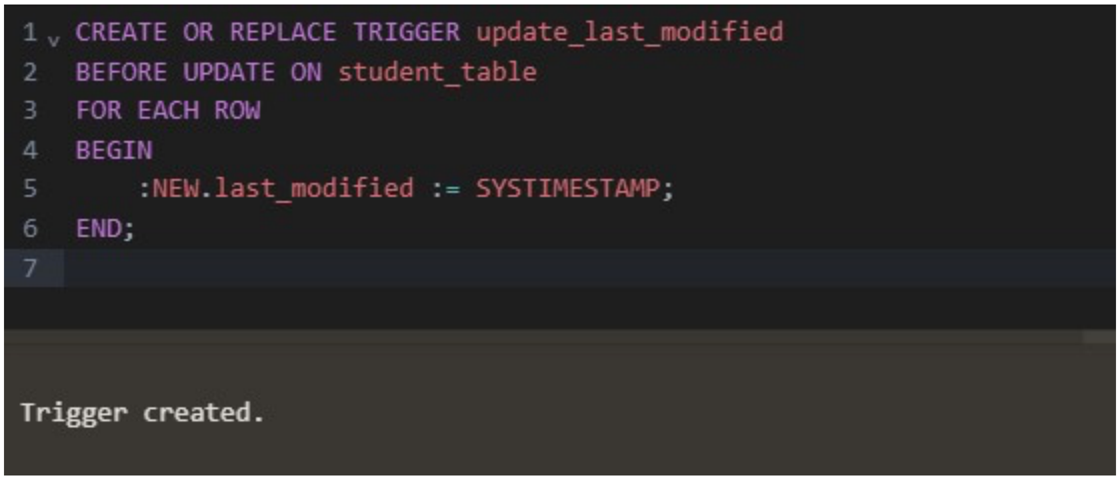
Output



```
SELECT calculate_square(5) AS square_result FROM dual;
```

```
SQUARE_RESULT
25
```

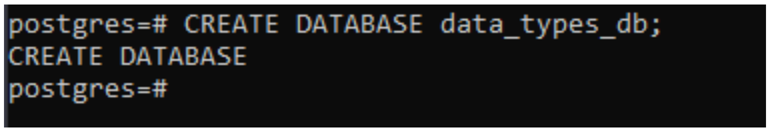
Triggers



```
CREATE OR REPLACE TRIGGER update_last_modified
BEFORE UPDATE ON student_table
FOR EACH ROW
BEGIN
    :NEW.last_modified := SYSTIMESTAMP;
END;
```

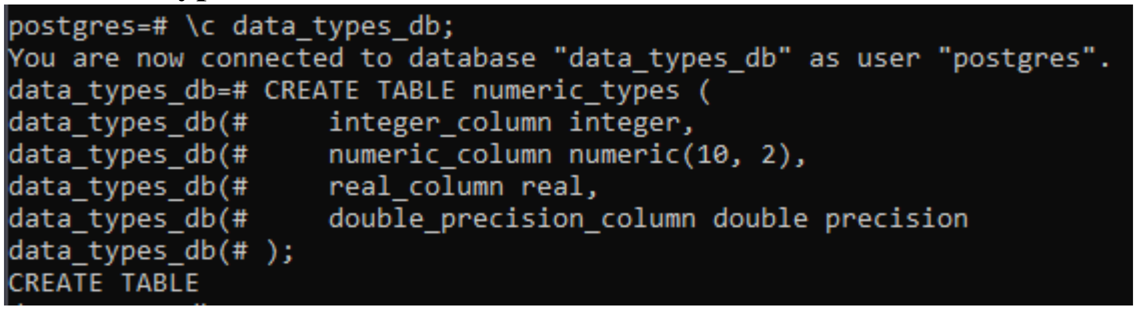
```
Trigger created.
```

3. Illustrate different data types in PostgreSQL



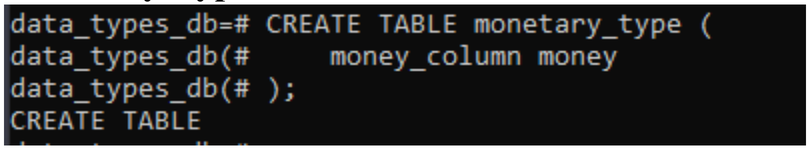
```
postgres=# CREATE DATABASE data_types_db;
CREATE DATABASE
postgres=#
```

- **Numeric Types**



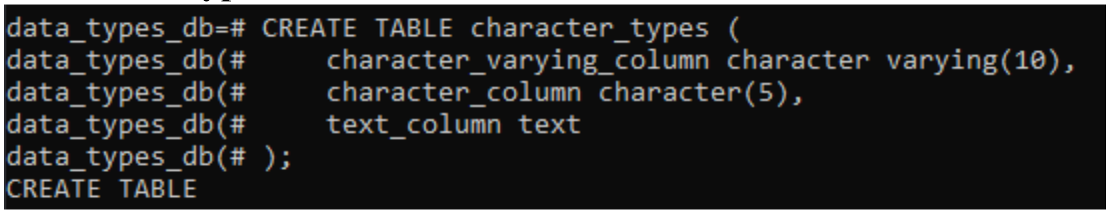
```
postgres=# \c data_types_db;
You are now connected to database "data_types_db" as user "postgres".
data_types_db=# CREATE TABLE numeric_types (
data_types_db(#     integer_column integer,
data_types_db(#     numeric_column numeric(10, 2),
data_types_db(#     real_column real,
data_types_db(#     double_precision_column double precision
data_types_db(# );
CREATE TABLE
```

- **Monetary Types**



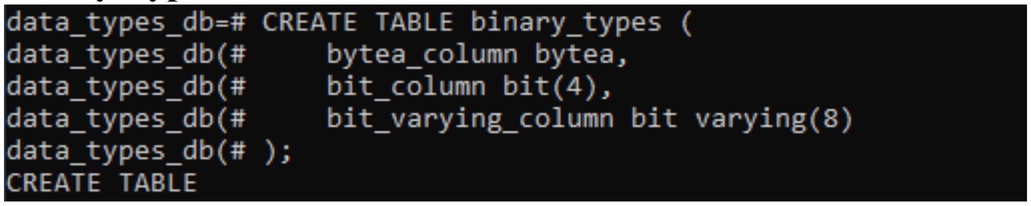
```
data_types_db=# CREATE TABLE monetary_type (
data_types_db(#     money_column money
data_types_db(# );
CREATE TABLE
```

- **Character Types**



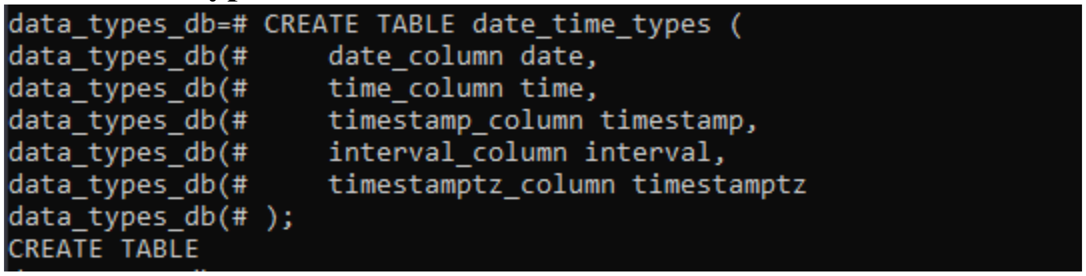
```
data_types_db=# CREATE TABLE character_types (
data_types_db(#     character_varying_column character varying(10),
data_types_db(#     character_column character(5),
data_types_db(#     text_column text
data_types_db(# );
CREATE TABLE
```

- **Binary Types**



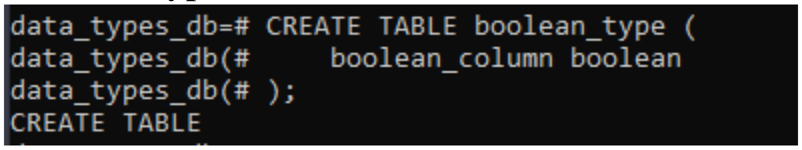
```
data_types_db=# CREATE TABLE binary_types (
data_types_db(#     bytea_column bytea,
data_types_db(#     bit_column bit(4),
data_types_db(#     bit_varying_column bit varying(8)
data_types_db(# );
CREATE TABLE
```

- **Date-Time Types**



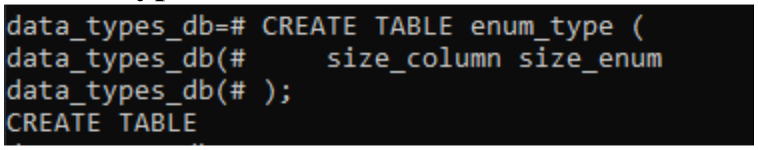
```
data_types_db=# CREATE TABLE date_time_types (
data_types_db(#     date_column date,
data_types_db(#     time_column time,
data_types_db(#     timestamp_column timestamp,
data_types_db(#     interval_column interval,
data_types_db(#     timestamptz_column timestamptz
data_types_db(# );
CREATE TABLE
```

- **Boolean Types**



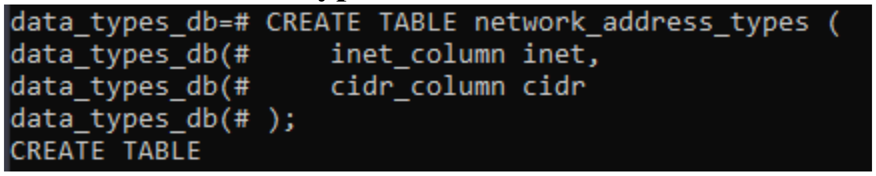
```
data_types_db=# CREATE TABLE boolean_type (
data_types_db(#     boolean_column boolean
data_types_db(# );
CREATE TABLE
```

- **Enum Types**



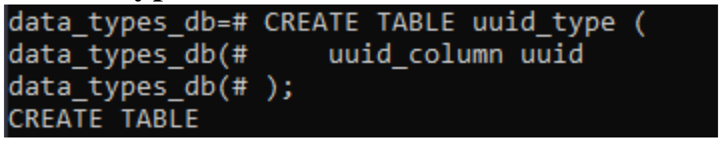
```
data_types_db=# CREATE TABLE enum_type (
data_types_db(#     size_column size_enum
data_types_db(# );
CREATE TABLE
```

- **Network Address Types**



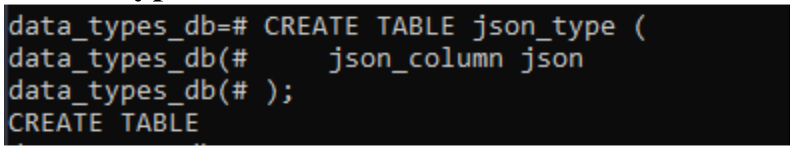
```
data_types_db=# CREATE TABLE network_address_types (
data_types_db(#     inet_column inet,
data_types_db(#     cidr_column cidr
data_types_db(# );
CREATE TABLE
```

- **UUID Types**



```
data_types_db=# CREATE TABLE uuid_type (
data_types_db(#     uuid_column uuid
data_types_db(# );
CREATE TABLE
```

- **JSON Types**

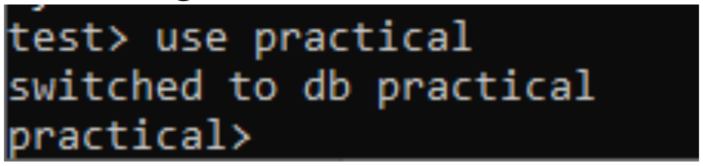


```
data_types_db=# CREATE TABLE json_type (
data_types_db(#     json_column json
data_types_db(# );
CREATE TABLE
```

4. Apply CRUD operations and retrieve data in NoSQL environment (Use MongoDB or any NoSQL database)

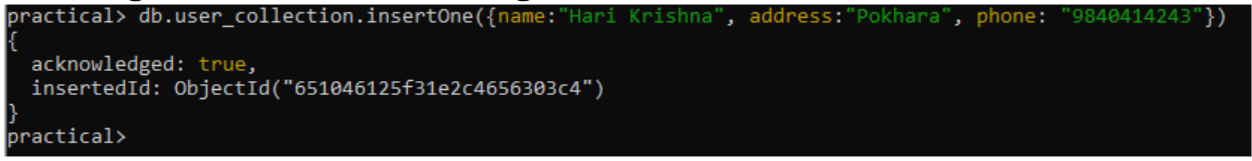
➔ Using MongoDB for performing CRUD operations

1. **Creating a Database**



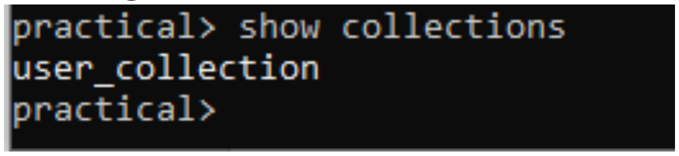
```
test> use practical
switched to db practical
practical>
```

2. **Creating a Collection and adding a Document for one User:**



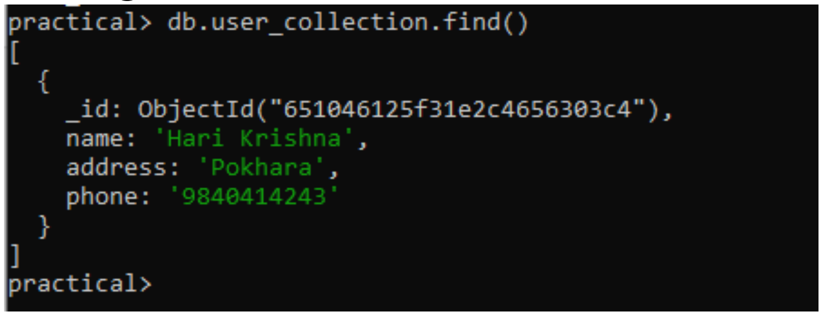
```
practical> db.user_collection.insertOne({name:"Hari Krishna", address:"Pokhara", phone: "9840414243"})
{
  acknowledged: true,
  insertedId: ObjectId("651046125f31e2c4656303c4")
}
practical>
```

3. **Showing the Current Collections**



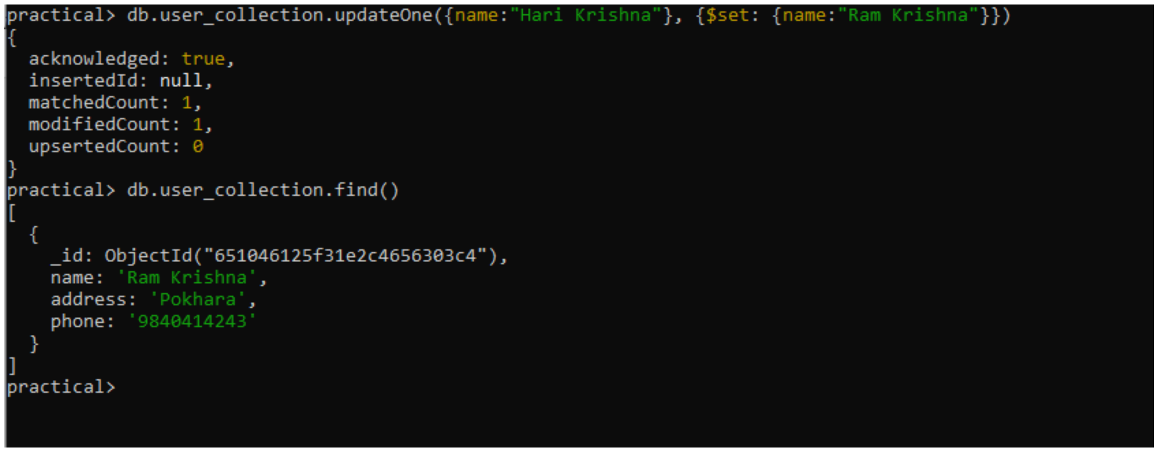
```
practical> show collections
user_collection
practical>
```

4. **Reading the Current Collection Data**



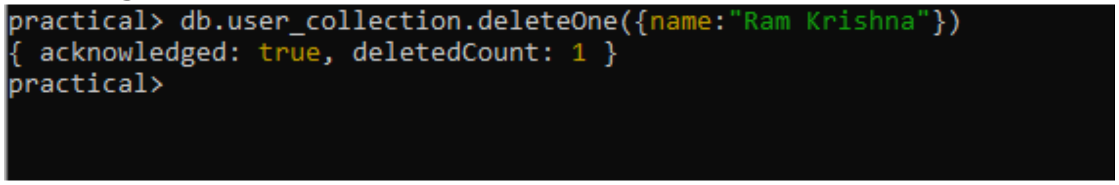
```
practical> db.user_collection.find()
[
  {
    _id: ObjectId("651046125f31e2c4656303c4"),
    name: 'Hari Krishna',
    address: 'Pokhara',
    phone: '9840414243'
  }
]
practical>
```

5. **Updating the Data**



```
practical> db.user_collection.updateOne({name:"Hari Krishna"}, {$set: {name:"Ram Krishna"}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
practical> db.user_collection.find()
[
  {
    _id: ObjectId("651046125f31e2c4656303c4"),
    name: 'Ram Krishna',
    address: 'Pokhara',
    phone: '9840414243'
  }
]
practical>
```

6. **Deleting the Data**



```
practical> db.user_collection.deleteOne({name:"Ram Krishna"})
{ acknowledged: true, deletedCount: 1 }
practical>
```

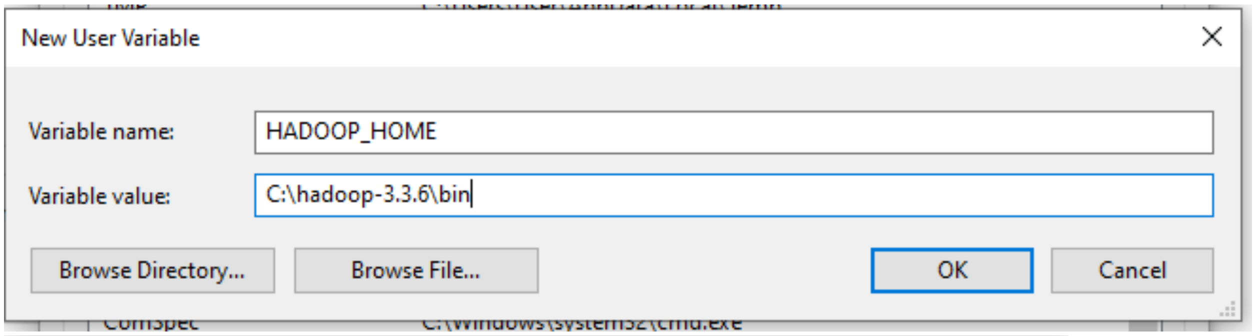
5. Understand the basic storage architecture of distributed file systems. Setup Apache Hadoop in your local machine

A distributed database file system is designed to manage and store data across multiple nodes or servers in a distributed computing environment. It combines aspects of both distributed databases and file systems to provide scalable and reliable storage for large volumes of data. The basic storage architecture of a distributed database file system typically includes the following components:

1. **Nodes and Server:** These are individual machines or servers that participate in the distributed file system. Each node has its own storage capacity and processing capabilities.
2. **Metadata Server:** In many distributed file systems, there is a central metadata server or a set of metadata servers that store information about the file system's structure and file attributes. This metadata includes details such as file names, directories, access permissions, and file locations.
3. **Data Storage:** The actual data files are distributed across multiple nodes in the system. Data can be broken down into smaller blocks or chunks, which are stored across different nodes for redundancy and load balancing.
4. **Replication and Redundancy:** To ensure data durability and fault tolerance, many distributed file systems replicate data across multiple nodes. This means that copies of the same data are stored on different servers. If one node fails, data can still be retrieved from other replicas.
5. **Access Control and Security:** Distributed file systems have mechanisms for access control and security to protect data from unauthorized access. This includes authentication, authorization, and encryption features.
6. **Load Balancing:** Load balancing techniques may be employed to evenly distribute data and query workloads across nodes to avoid overloading any single server.
7. **Data Recovery and Backup:** Distributed file systems often include mechanisms for data recovery and backup, allowing administrators to restore data in case of data corruption or node failures.
8. **Monitoring and Management Tools:** To monitor the health and performance of the distributed file system, management tools and monitoring systems are essential. These tools help administrators diagnose issues, optimize performance, and ensure the system is operating as expected.

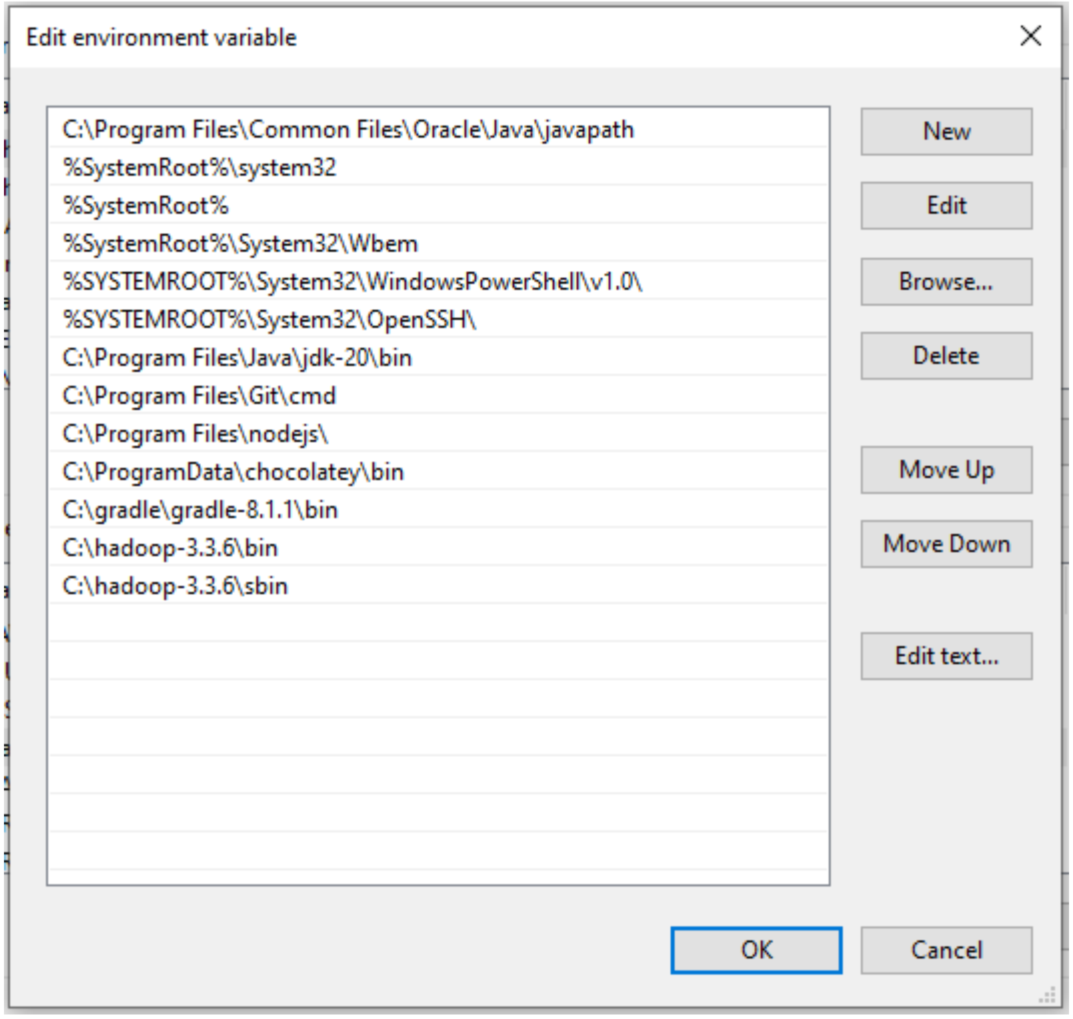
Apache Hadoop Setup:

1. First configure the Hadoop env file by setting up the Java path.
2. Add the Hadoop path to Environment Variable.



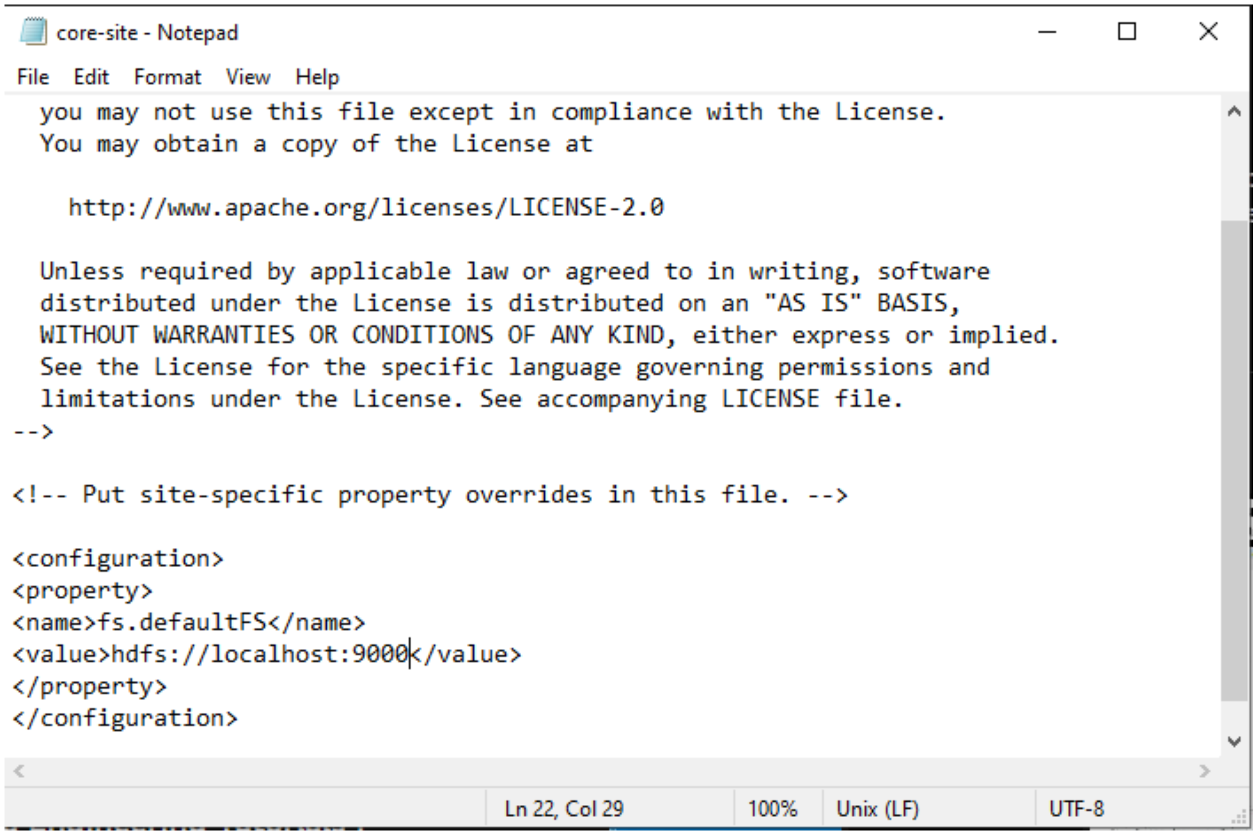
Variable name: HADOOP_HOME

Variable value: C:\hadoop-3.3.6\bin



```
C:\Program Files\Common Files\Oracle\Java\javapath
%SystemRoot%\system32
%SystemRoot%
%SystemRoot%\System32\Wbem
%SYSTEMROOT%\System32\WindowsPowerShell\v1.0\
%SYSTEMROOT%\System32\OpenSSH\
C:\Program Files\Java\jdk-20\bin
C:\Program Files\Git\cmd
C:\Program Files\nodejs\
C:\ProgramData\chocolatey\bin
C:\gradle\gradle-8.1.1\bin
C:\hadoop-3.3.6\bin
C:\hadoop-3.3.6\sbin
```

3. Configure core-site document



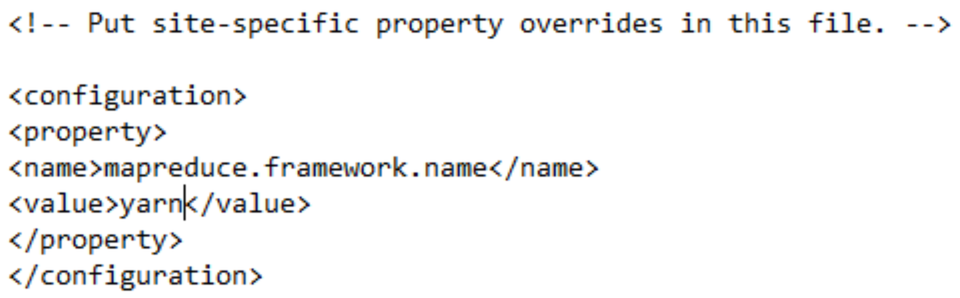
```
  you may not use this file except in compliance with the License.
  You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License. See accompanying LICENSE file.
-->

<!-- Put site-specific property overrides in this file. -->

<configuration>
<property>
<name>fs.defaultFS</name>
<value>hdfs://localhost:9000</value>
</property>
</configuration>
```

4. Now configure httpfs-site file or hdfs-site, either one will work.
   4.1 Create one folder called data and inside the create two more folder namenode and datanode simultaneously.

   4.2 Add the folder location to the configurable file httpfs-site or hdfs-site

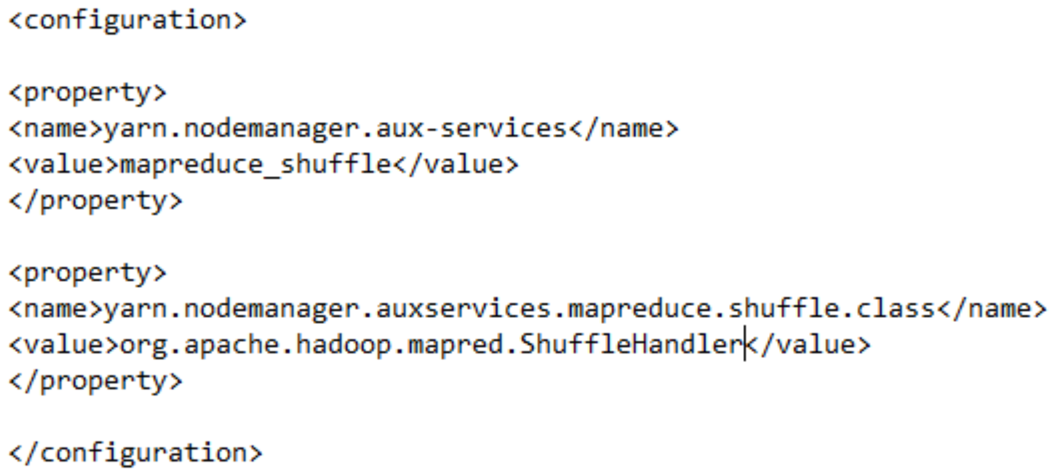
```
<configuration>
<property>
<name>dfs.replication</name>
<value>1</value>
</property>
<property>
<name>dfs.namenode.name.dir</name>
<value>C:\hadoop-3.3.6\data\namenode</value>
</property>
<property>
<name>dfs.datanode.data.dir</name>
<value>C:\hadoop-3.3.6\data\datanode</value>
</property>
</configuration>
```

5. Now configure the mapred-site file



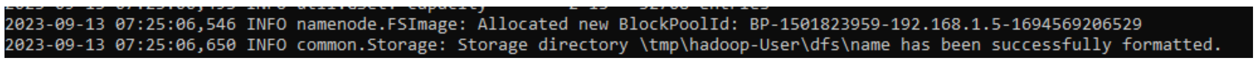
```
<!-- Put site-specific property overrides in this file. -->

<configuration>
<property>
<name>mapreduce.framework.name</name>
<value>yarn</value>
</property>
</configuration>
```

6. We need to configure yarn-site file



```
<configuration>

<property>
<name>yarn.nodemanager.aux-services</name>
<value>mapreduce_shuffle</value>
</property>

<property>
<name>yarn.nodemanager.auxservices.mapreduce.shuffle.class</name>
<value>org.apache.hadoop.mapred.ShuffleHandler</value>
</property>

</configuration>
```

7. Format the hdfs namenode

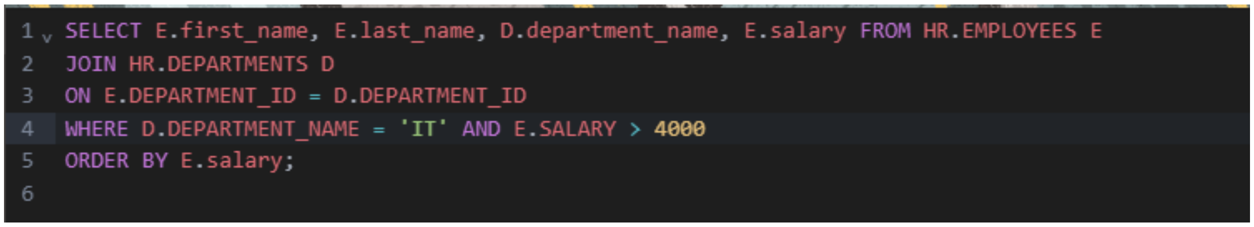


```
2023-09-13 07:25:06,546 INFO namenode.FSImage: Allocated new BlockPoolId: BP-1501823959-192.168.1.5-1694569206529
2023-09-13 07:25:06,650 INFO common.Storage: Storage directory \tmp\hadoop-User\dfs\name has been successfully formatted.
```

8. Start namenode and datanode by using command **start-dfs.cmd**
9. Start yarn by using command **start-yarn.cmd**

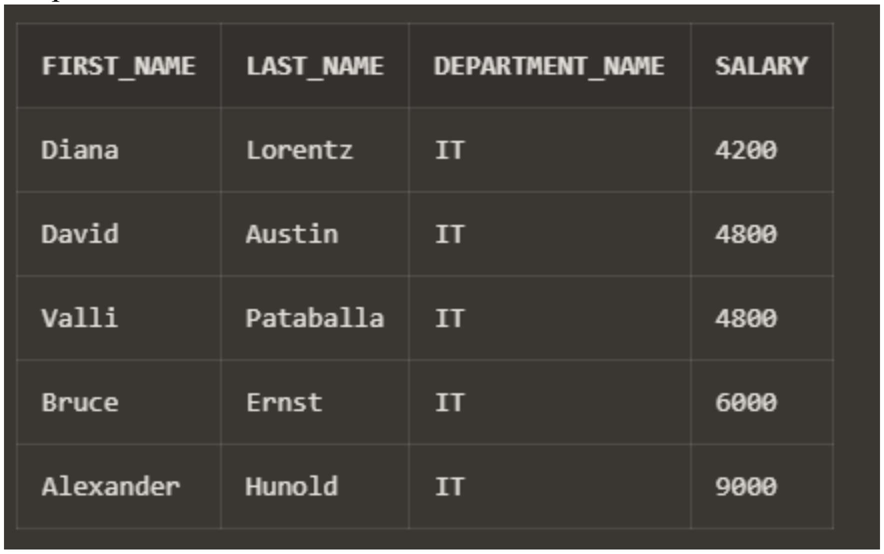
6. Distributed Horizontal Fragmentation

Divide Employees table with IT and Sales department in different fragments



```sql
SELECT E.first_name, E.last_name, D.department_name, E.salary FROM HR.EMPLOYEES E
JOIN HR.DEPARTMENTS D
ON E.DEPARTMENT_ID = D.DEPARTMENT_ID
WHERE D.DEPARTMENT_NAME = 'IT' AND E.SALARY > 4000
ORDER BY E.salary;
```

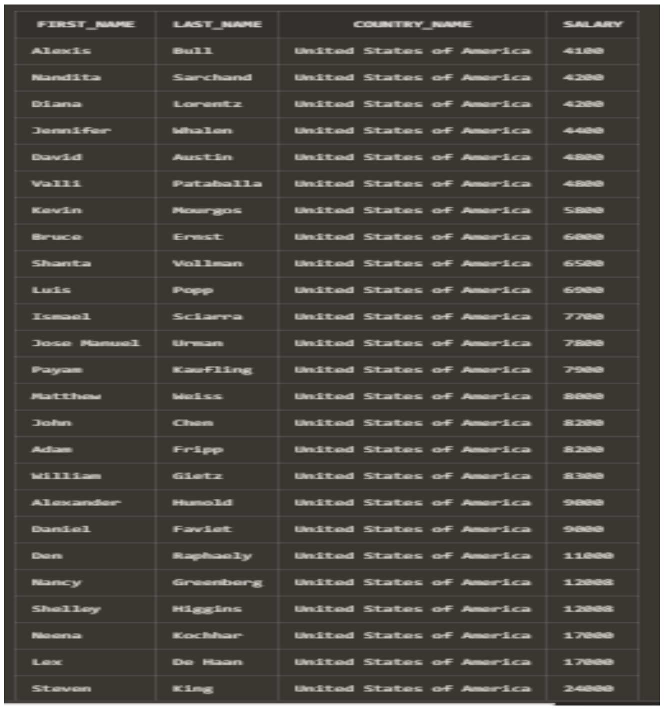
Output:



FIRST_NAME	LAST_NAME	DEPARTMENT_NAME	SALARY
Diana	Lorentz	IT	4200
David	Austin	IT	4800
Valli	Pataballa	IT	4800
Bruce	Ernst	IT	6000
Alexander	Hunold	IT	9000

Find all employees who works on department located at country 'United States of America' and have salary>4000

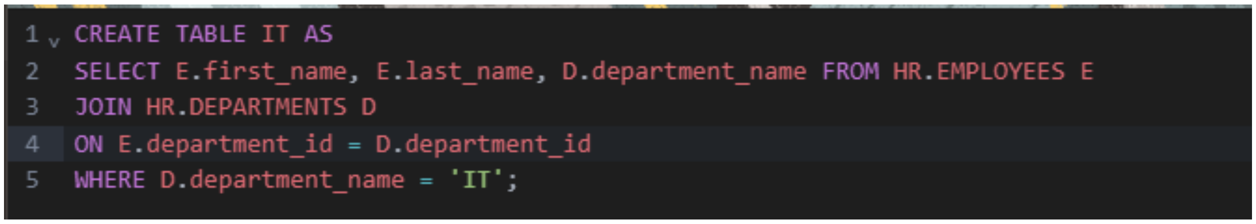
Output:



FIRST_NAME	LAST_NAME	COUNTRY_NAME	SALARY
Alexis	Bull	United States of America	4100
Nandita	Sarchand	United States of America	4200
Diana	Lorentz	United States of America	4200
Jennifer	Whalen	United States of America	4400
David	Austin	United States of America	4800
Valli	Pataballa	United States of America	4800
Kevin	Mourgos	United States of America	5800
Bruce	Ernst	United States of America	6000
Shanta	Vollman	United States of America	6500
Luis	Popp	United States of America	6900
Ismael	Sciarra	United States of America	7700
Jose Manuel	Urman	United States of America	7800
Payam	Kaufling	United States of America	7900
Matthew	Weiss	United States of America	8000
John	Chen	United States of America	8200
Adam	Fripp	United States of America	8200
William	Gietz	United States of America	8300
Alexander	Hunold	United States of America	9000
Daniel	Faviet	United States of America	9000
Den	Raphaely	United States of America	11000
Nancy	Greenberg	United States of America	12008
Shelley	Higgins	United States of America	12008
Neena	Kochhar	United States of America	17000
Lex	De Haan	United States of America	17000
Steven	King	United States of America	24000

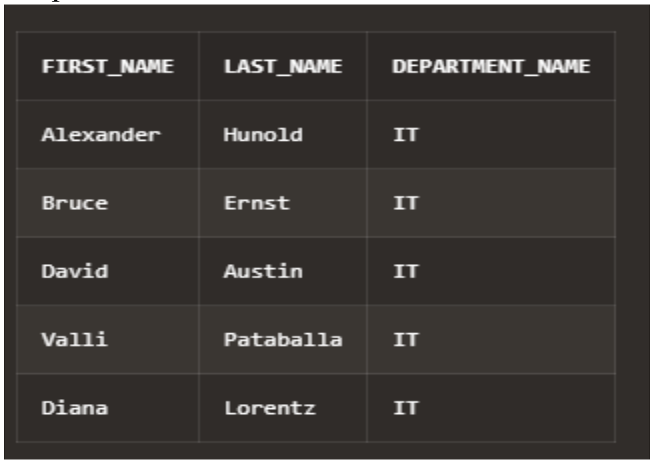
7. Distributed Vertical Fragmentation

Divide Employees table with IT and Sales department in different fragments

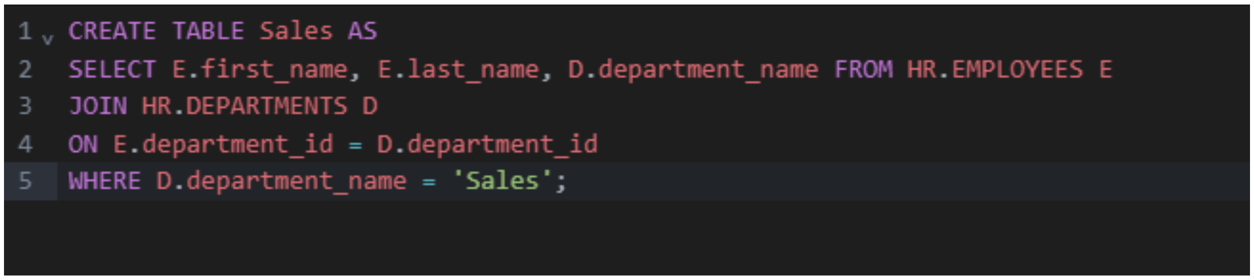


```sql
CREATE TABLE IT AS
SELECT E.first_name, E.last_name, D.department_name FROM HR.EMPLOYEES E
JOIN HR.DEPARTMENTS D
ON E.department_id = D.department_id
WHERE D.department_name = 'IT';
```

Output:

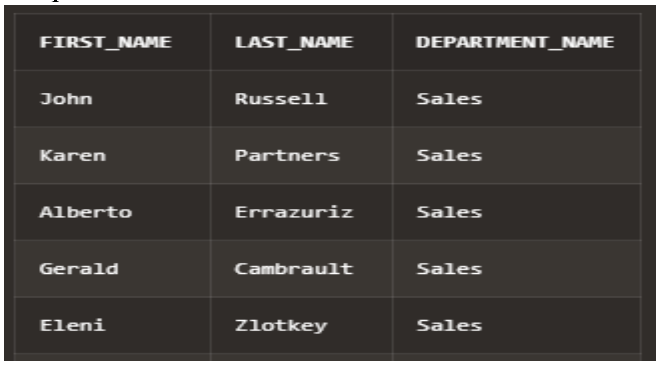


FIRST_NAME	LAST_NAME	DEPARTMENT_NAME
Alexander	Hunold	IT
Bruce	Ernst	IT
David	Austin	IT
Valli	Pataballa	IT
Diana	Lorentz	IT



```sql
CREATE TABLE Sales AS
SELECT E.first_name, E.last_name, D.department_name FROM HR.EMPLOYEES E
JOIN HR.DEPARTMENTS D
ON E.department_id = D.department_id
WHERE D.department_name = 'Sales';
```

Output:



FIRST_NAME	LAST_NAME	DEPARTMENT_NAME
John	Russell	Sales
Karen	Partners	Sales
Alberto	Errazuriz	Sales
Gerald	Cambrault	Sales
Eleni	Zlotkey	Sales

8. Using prolog, perform the following tasks for the Family Relations:

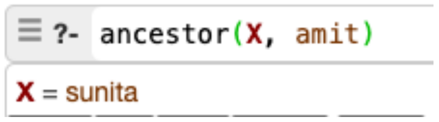
i. Define facts to represent parent-child relationships in the family tree.

```
parent(rajesh, sunita).
parent(rajesh, arjun).
parent(sunita, rina).
parent(sunita, amit).
parent(arjun, neha).
parent(arjun, rohan).
parent(rina, vishal).
parent(amit, preeti).
parent(neha, karan).
parent(rohan,sneha).

female(sunita).
female(rina).
female(neha).
female(preeti).
female(sneha).

male(rajesh).
male(arjun).
male(amit).
male(vishal).
male(karan).
male(rohan).
```

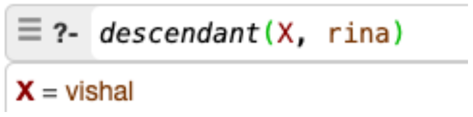
ii. Define facts to represent parent-child relationships in the family tree.

```
% ancestor relationship
ancestor(X, Y) :- parent(X, Y).
ancestor(X, Y) :- parent(X, Z), ancestor(Z, Y).

% descendant relationship
descendant(X, Y) :- parent(Y, X).
descendant(X, Y) :- parent(Z, X), descendant(Z, Y).
```

iii. Write queries to find all ancestors and descendants of a given person.



```
?- ancestor(X, amit)
X = sunita
```

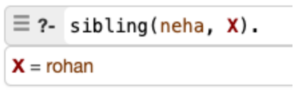


```
?- descendant(X, rina)
X = vishal
```

iv. Define rules to represent sibling relationships.

sibling(X,Y):-parent(P,X),parent(P,Y),X\=Y.

v. Write queries to find all siblings of a person.

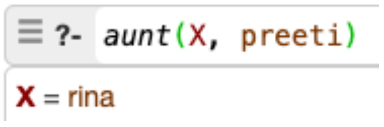


```
?- sibling(neha, X).
X = rohan
```

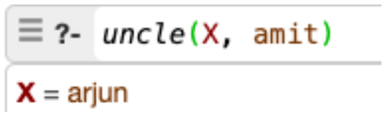
vi. Define rules to represent the aunt and uncle relationships.

```
% Define aunt relationship
aunt(Aunt, NieceNephew) :-
    parent(Parent, NieceNephew),
    sibling(Aunt, Parent),
    female(Aunt).

% Define uncle relationship
uncle(Uncle, NieceNephew) :-
    parent(Parent, NieceNephew),
    sibling(Uncle, Parent),
    male(Uncle).
```

vii. Write queries to find aunts and uncles of a person.



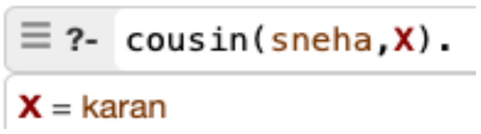
```
?- aunt(X, preeti)
X = rina
```


```
?- uncle(X, amit)
X = arjun
```

viii. Define rules to represent the aunt and uncle relationships.

```
% Define cousin relationship
cousin(Cousin, Person) :-
    parent(Parent1, Cousin),
    parent(Parent2, Person),
    sibling(Parent1, Parent2),
    \+ sibling(Cousin, Person),
    Cousin \= Person.
```

ix. Write a query to find all cousins of a given person.


```
?- cousin(sneha,X).
X = karan
```