EXP No.10

Experiment to implement different Fragmentation techniques in a Distributed Database, Check the correctness criteria, and execute distributed queries on the same.(SQL Server Linked Server /implementation using Java /Python)

🔹 Fragmentation in Distributed Databases

Fragmentation is a technique used in **distributed databases** to improve **performance, availability, and manageability** by dividing a database into smaller pieces called **fragments**. Each fragment can then be stored at different sites (nodes) in the network.

Purpose of fragmentation:

1. Reduce **data transfer costs** by storing data closer to where it is needed.
2. Improve **query performance** by accessing only relevant fragments.
3. Enhance **availability** and **parallelism**.

1️⃣Types of Fragmentation

Horizontal → split rows

Vertical → split columns

Hybrid → combine both

Key principles → completeness, reconstructability, disjointness

1. Horizontal Fragmentation

- Divides a table **row-wise**.
- Each fragment contains a subset of rows based on a **selection predicate**.

- Example:
  Table `Employees` → Fragment by department:
  - Fragment 1: `Employees` where `Dept = 'HR'`
  - Fragment 2: `Employees` where `Dept = 'Sales'`

Use case: Queries often access only specific rows, e.g., queries for a specific region or department.

---

2. Vertical Fragmentation

- Divides a table **column-wise**.
- Each fragment contains a subset of columns plus the **primary key** to allow reconstruction.
- Example:
  Table `Employees` → Fragment by columns:
  - Fragment 1: `EmpID, Name, Dept`
  - Fragment 2: `EmpID, Salary, Address`

Use case: Some applications only need part of the table (e.g., HR system only accesses salary, others access contact info).

---

3. Mixed / Hybrid Fragmentation

- Combines **horizontal + vertical** fragmentation.
- Example:
  1. First split `Employees` horizontally by department.
  2. Then split each fragment vertically into sensitive vs non-sensitive columns.

Use case: Large distributed databases needing fine-grained control for performance, security, or replication.

2️⃣ Fragmentation Principles

1. **Completeness:** All fragments together should represent the **entire original table**.
2. **Reconstructability / Reconstructability:** It must be possible to reconstruct the original table from fragments using **UNION (horizontal)** or **JOIN (vertical)**.
3. **Disjointness:** Fragments should ideally not **overlap** unless replication is used.

3️⃣ Benefits of Fragmentation

- Improved **query performance** (access only relevant fragments).
- Reduced **communication cost** (data stored closer to users).
- Better **data security** (sensitive data can be separated).
- Easier **parallel processing** and **load balancing**.

1️⃣ Setup: Original Table

```sql
-- Step 1: Create database and original table
DROP DATABASE IF EXISTS distributed_demo;
CREATE DATABASE distributed_demo;
USE distributed_demo;

CREATE TABLE Employees (
  EmpID INT PRIMARY KEY,
  Name VARCHAR(50),
  Dept VARCHAR(50),
  Salary DECIMAL(10,2),
  Address VARCHAR(100)
);
```

```sql
-- Insert sample data
INSERT INTO Employees (EmpID, Name, Dept, Salary, Address) VALUES
(1,'Amit','HR',50000,'Delhi'),
(2,'Divya','Sales',45000,'Mumbai'),
(3,'Rahul','IT',120000,'Bangalore'),
(4,'Sneha','HR',55000,'Delhi'),
(5,'Rohit','Sales',47000,'Mumbai');
```

Check data:

sql   Copy code

```sql
SELECT * FROM Employees;
```

Expected Output:

EmpID	Name	Dept	Salary	Address
1	Amit	HR	50000	Delhi
2	Divya	Sales	45000	Mumbai
3	Rahul	IT	120000	Bangalore
4	Sneha	HR	55000	Delhi
5	Rohit	Sales	47000	Mumbai

2️⃣ Horizontal Fragmentation

Fragment by Department (`Dept = 'HR'`, `Dept = 'Sales'`):

```
-- HR Fragment
CREATE TABLE Employees_HR AS
SELECT * FROM Employees WHERE Dept='HR';

-- Sales Fragment
CREATE TABLE Employees_Sales AS
SELECT * FROM Employees WHERE Dept='Sales';

-- IT Fragment (optional)
CREATE TABLE Employees_IT AS
SELECT * FROM Employees WHERE Dept='IT';
```

Check fragments:

```sql
SELECT * FROM Employees_HR;
SELECT * FROM Employees_Sales;
SELECT * FROM Employees_IT;
```

Expected Output:

`Employees_HR`

EmpID	Name	Dept	Salary	Address
1	Amit	HR	50000	Delhi
4	Sneha	HR	55000	Delhi

`Employees_Sales`

EmpID	Name	Dept	Salary	Address
2	Divya	Sales	45000	Mumbai
5	Rohit	Sales	47000	Mumbai

`Employees_IT`

EmpID	Name	Dept	Salary	Address
3	Rahul	IT	120000	Bangalore

3 Vertical Fragmentation

Split columns while keeping `EmpID` as primary key for reconstruction

```
-- Fragment 1: EmpID, Name, Dept
CREATE TABLE Employees_V1 AS
SELECT EmpID, Name, Dept FROM Employees;

-- Fragment 2: EmpID, Salary, Address
CREATE TABLE Employees_V2 AS
SELECT EmpID, Salary, Address FROM Employees;
```

Check fragments:

```sql
SELECT * FROM Employees_V1;
SELECT * FROM Employees_V2;
```

Expected Output:

`Employees_V1`

EmpID	Name	Dept
1	Amit	HR
2	Divya	Sales
3	Rahul	IT
4	Sneha	HR
5	Rohit	Sales

`Employees_V2`

EmpID	Salary	Address
1	50000	Delhi
2	45000	Mumbai
3	120000	Bangalore
4	55000	Delhi
5	47000	Mumbai

4 Hybrid Fragmentation

Example: Horizontal by Dept='HR' vs Dept='Sales', then vertical for each fragment

```
-- HR Horizontal + Vertical
CREATE TABLE Employees_HR_V1 AS SELECT EmpID, Name FROM Employees WHERE Dept='HR';
CREATE TABLE Employees_HR_V2 AS SELECT EmpID, Salary FROM Employees WHERE Dept='HR';

-- Sales Horizontal + Vertical
CREATE TABLE Employees_Sales_V1 AS SELECT EmpID, Name FROM Employees WHERE Dept='Sales';
CREATE TABLE Employees_Sales_V2 AS SELECT EmpID, Salary FROM Employees WHERE Dept='Sales';
```

5️⃣ Check Correctness Criteria

1. Completeness: Union all horizontal fragments = original table

```sql
SELECT * FROM Employees_HR
UNION ALL
SELECT * FROM Employees_Sales
UNION ALL
SELECT * FROM Employees_IT;
```

- Must match original `Employees` table.

2. Reconstructability: Join vertical fragments on `EmpID`

```sql
SELECT V1.EmpID, V1.Name, V1.Dept, V2.Salary, V2.Address
FROM Employees_V1 V1
JOIN Employees_V2 V2 ON V1.EmpID = V2.EmpID;
```

- Should reconstruct original table perfectly.

3. Disjointness: Ensure horizontal fragments do not overlap

```sql
SELECT COUNT(*) FROM Employees_HR H
JOIN Employees_Sales S ON H.EmpID = S.EmpID;
```

- Expected: 0 → no overlapping rows.

6️⃣Execute Distributed Queries on Fragments

Example Query: Get names and salaries of HR employees

```sql
SELECT H.EmpID, H.Name, V2.Salary
FROM Employees_HR H
JOIN Employees_V2 V2 ON H.EmpID = V2.EmpID;
```

Expected Output:

EmpID	Name	Salary
1	Amit	50000
4	Sneha	55000

Example Query: Average salary by department (using horizontal fragments)

```sql
SELECT 'HR' AS Dept, AVG(Salary) AS AvgSalary FROM Employees_HR
UNION ALL
SELECT 'Sales', AVG(Salary) FROM Employees_Sales
UNION ALL
SELECT 'IT', AVG(Salary) FROM Employees_IT;
```

Expected Output:

Dept	AvgSalary
HR	52500
Sales	46000
IT	20000