**Distributed Database Project – POS System**

**Objective**

To design and implement a Distributed Database System that demonstrates concepts of data fragmentation, replication, concurrency control, query processing, fault tolerance, and recovery using PostgreSQL.

**Project Deliverables**

**Each group must submit:**

1. **Source Code File** – All DDL (tables, schema, fragments, replication rules) and DML (queries, procedures, triggers, transactions) scripts in PostgreSQL.
2. **Complete Project Report** – Includes diagrams, ER models, fragmentation schemas, and test cases.
3. **Presentation (12–15 slides)** – Covers problem definition, database design, DDB aspects, implementation screenshots, and results.

**1. Introduction & Problem Statement**

**Background:** The POS system manages customers, products, stores, orders, and inventory.  
**Need for Distributed Database:**

* Multi-location stores require real-time data replication.
* Centralized DB could become a bottleneck and single point of failure.
* Distributed system ensures high availability and scalability.

**2. System Design & Architecture**

**ER Diagram & Schema Design**

* Entities: Customers, Stores, Products, Orders, Order\_Items, Inventory, Payments, Cities, Categories.
* Relationships:
  + Stores belong to Cities.
  + Orders linked to Customers and Stores.
  + Order\_Items linked to Orders and Products.
  + Inventory per store per product.

**Fragmentation & Replication**

* **Horizontal Fragmentation:** Customers and Stores replicated to store DBs based on city.
* **Replication:**
  + Customers and Stores replicated to respective store databases using triggers + dblink.
  + Replication log tracks success/failure of replication.

**Concurrency Control**

* Orders processed with ProcessOrder procedure.
* Inventory rows locked with FOR UPDATE to prevent race conditions.

**3. Implementation in PostgreSQL**

**Central Database (pos\_central)**

* Tables: Cities, Categories, Products, Stores, Customers, Order\_Mapping, Replication\_Log.
* Roles: central\_admin (full), dblink\_user (restricted).

**Store Databases (karachi\_db, lahore\_db)**

* Tables: Customers, Stores, Orders, Order\_Items, Payments, Inventory.

**Replication Triggers**

* Customers and Stores replicated automatically with AFTER INSERT OR UPDATE triggers.
* retry\_failed\_replications() procedure ensures fault tolerance.

**Stored Procedures**

* ProcessOrder – Inserts orders, updates inventory, processes payment, ensures concurrency control.

**4. Query Processing & Optimization**

* Distributed query to calculate total sales across stores:

WITH karachi\_sales AS (

SELECT SUM(quantity\*price) AS total

FROM dblink('dbname=karachi\_db user=dblink\_user password=dblink123',

'SELECT quantity, price FROM Order\_Items') AS t(quantity INT, price NUMERIC)

),

lahore\_sales AS (

SELECT SUM(quantity\*price) AS total

FROM dblink('dbname=lahore\_db user=dblink\_user password=dblink123',

'SELECT quantity, price FROM Order\_Items') AS t(quantity INT, price NUMERIC)

)

SELECT COALESCE(k.total,0)+COALESCE(l.total,0) AS total\_sales

FROM karachi\_sales k, lahore\_sales l;

* Query execution analyzed with EXPLAIN ANALYZE.

**5. Fault Tolerance, Security & Recovery**

* **Backup/Recovery**
  + pg\_dump and pg\_restore for central & store DBs.
* **Security**
  + Roles: central\_admin (all privileges), dblink\_user (read/write access only for replication).
* **Fault Tolerance**
  + Failed replications logged in Replication\_Log.
  + Automatic retry procedure to ensure data consistency.

**6. Results, Conclusion & Future Enhancements**

* **Performance Evaluation:** Orders processed concurrently without inventory conflicts.
* **Lessons Learned:** Importance of replication, triggers, and logging in distributed systems.
* **Future Scope:**
  + Include Products and Orders replication.
  + Real-time analytics dashboard.
  + Multi-region deployment for scalability.

**Key Distributed Database Concepts Demonstrated**

* Fragmentation (Horizontal)
* Replication
* Distributed Query Processing
* Concurrency Control
* Fault Tolerance & Recovery