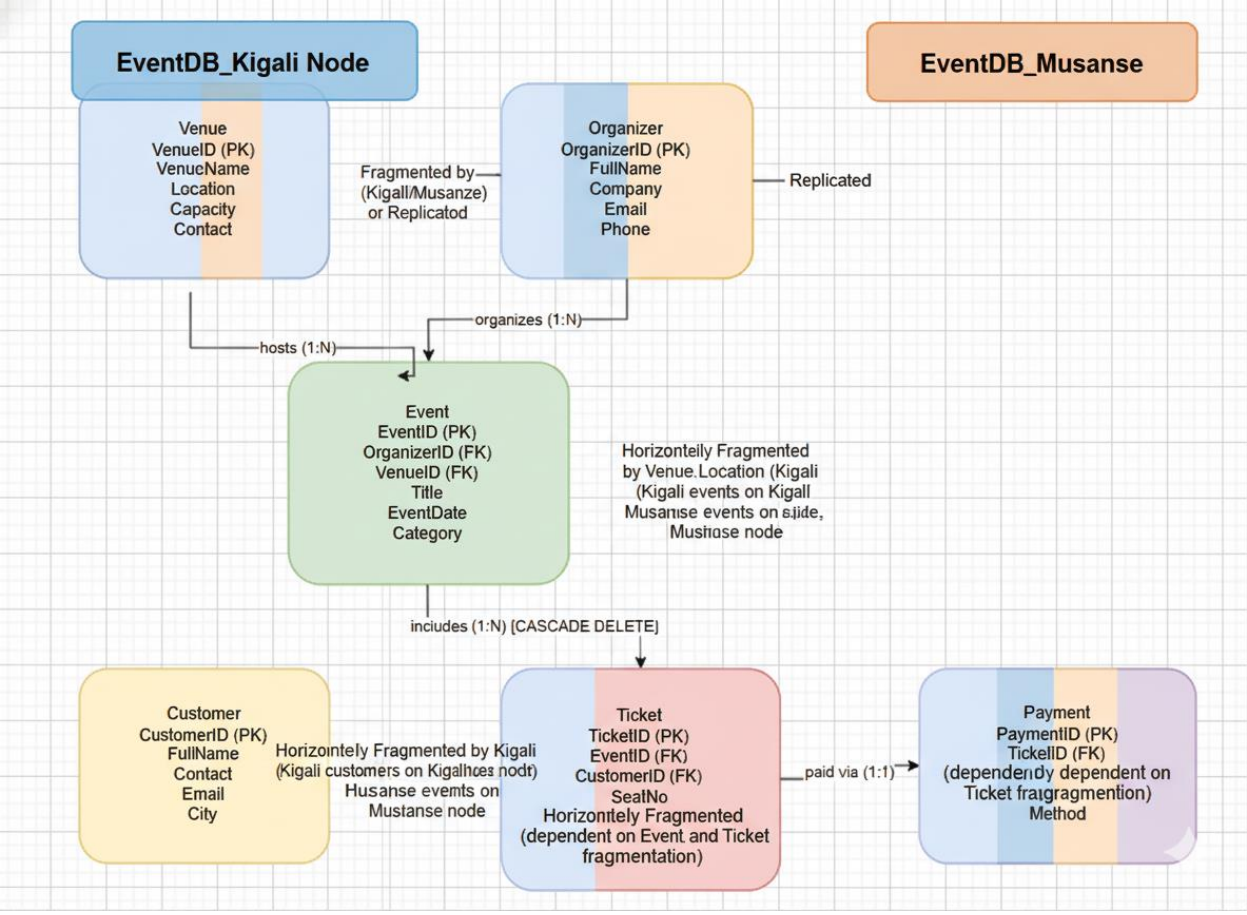
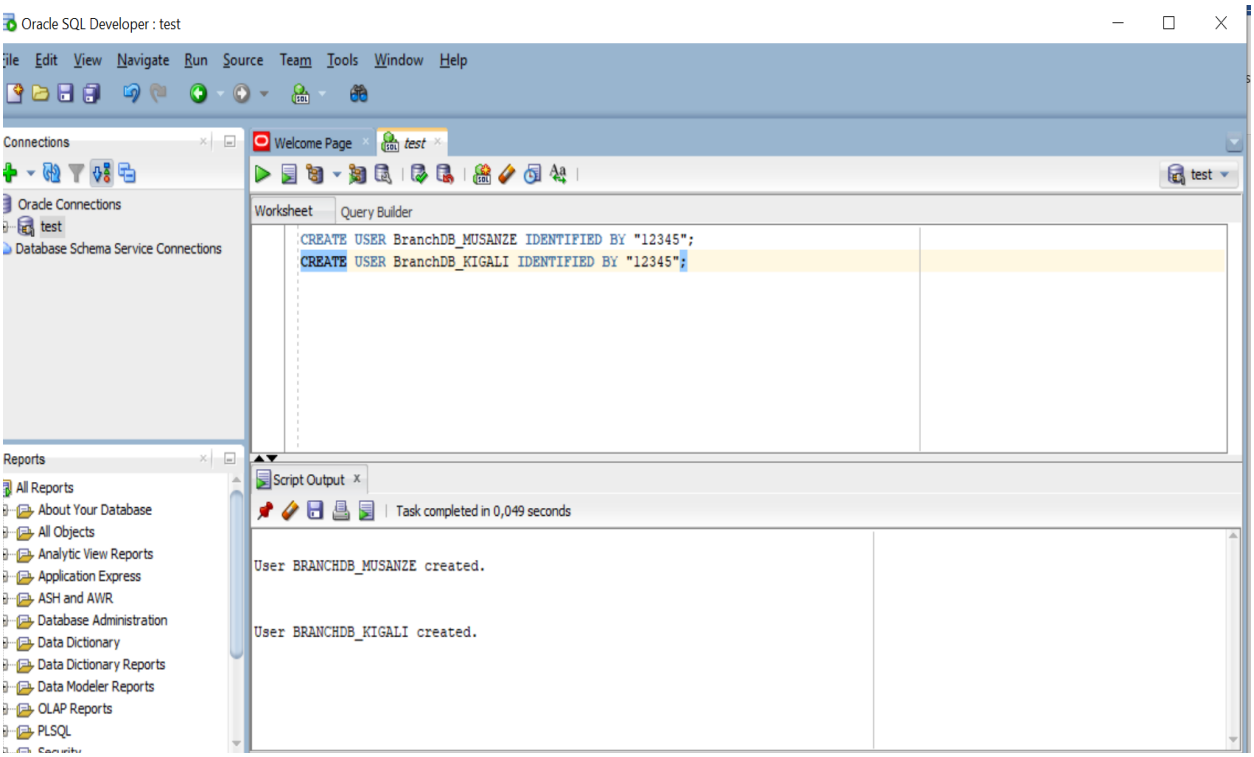
Topic: Parallel and Distributed Databases

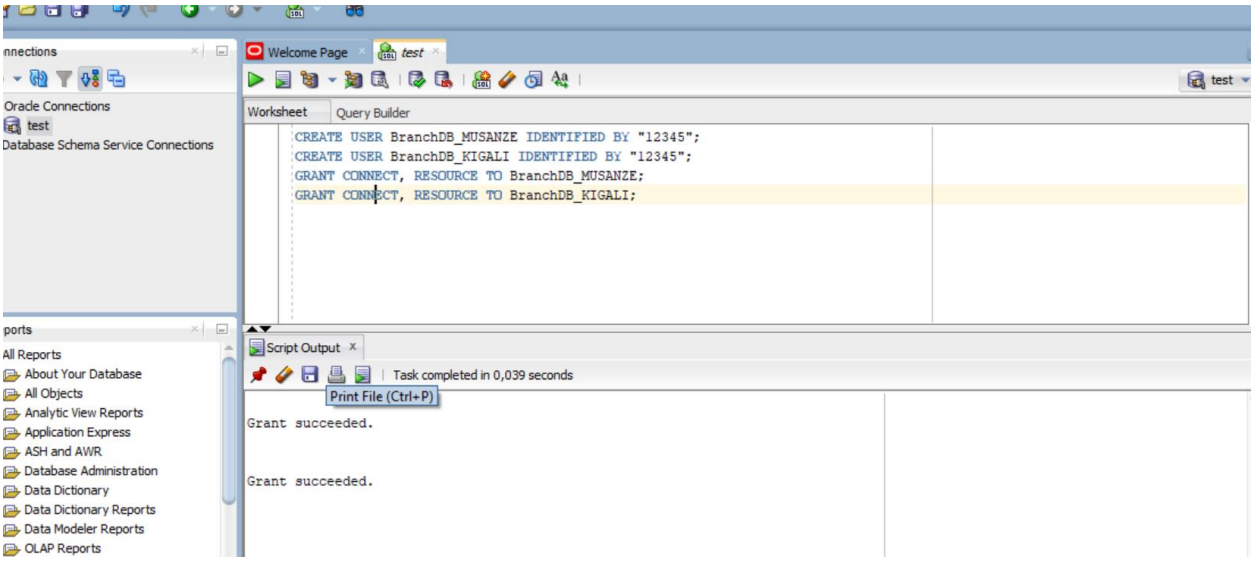
Entity Relationship Diagram (nod_a and node_b)



Q1..CREATE USER BranchDB_MUSANZE IDENTIFIED BY "12345";

CREATE USER BranchDB_KIGALI IDENTIFIED BY "12345";



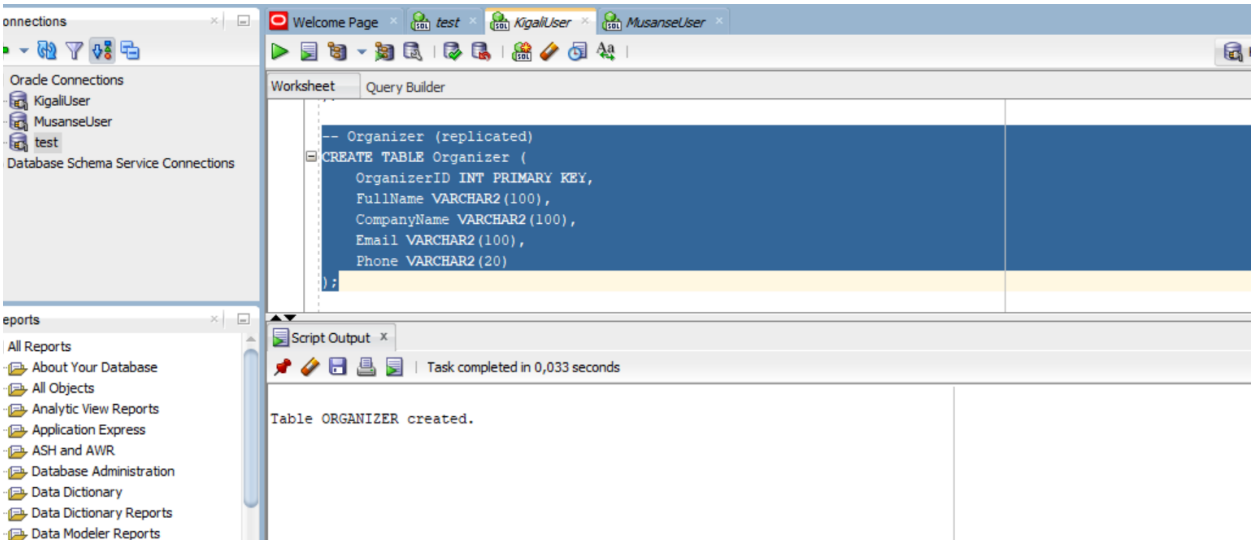


GRANT CONNECT, RESOURCE TO BranchDB_MUSANZE;

GRANT CONNECT, RESOURCE TO BranchDB_KIGALI;

Creating tables

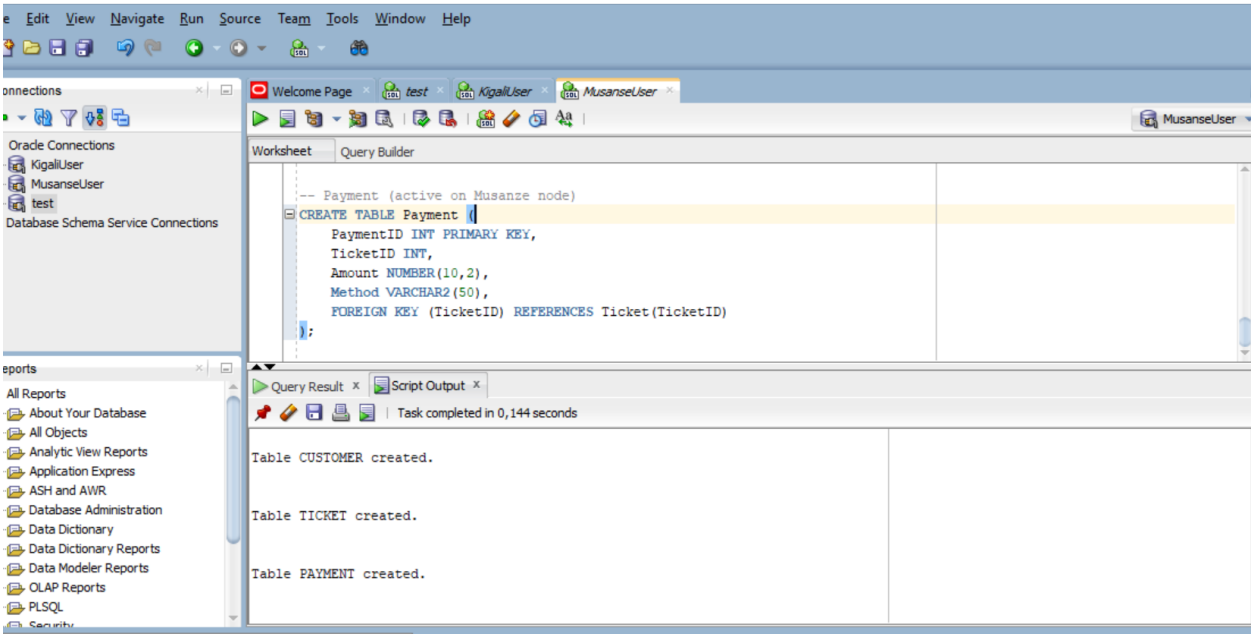
```
CREATE TABLE Organizer (
    OrganizerID INT PRIMARY KEY,
    FullName VARCHAR2(100),
    CompanyName VARCHAR2(100),
    Email VARCHAR2(100),
    Phone VARCHAR2(20)
);
```



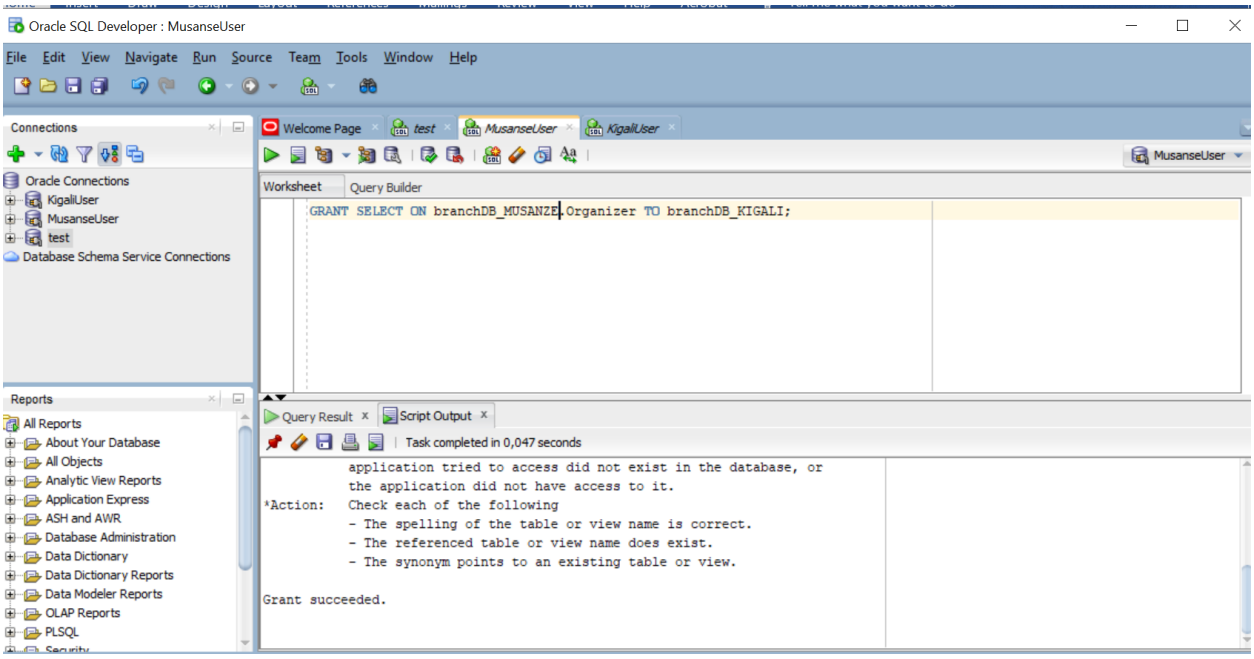

```
-- Venue (replicated, but filtered for Musanze)
CREATE TABLE Venue (
    VenueID INT PRIMARY KEY,
    VenueName VARCHAR2(100),
    VenueLocation VARCHAR2(100) CHECK (VenueLocation = 'Musanze')
);

-- Organizer (replicated)
CREATE TABLE Organizer (
    OrganizerID INT PRIMARY KEY,
    FullName VARCHAR2(100),
    CompanyName VARCHAR2(100),
    Email VARCHAR2(100),
    Phone VARCHAR2(20)
```

```
);

-- Event (fragmented by VenueLocation = Musanze)
CREATE TABLE Event (
    EventID INT PRIMARY KEY,
    OrganizerID INT,
    VenueID INT,
    Title VARCHAR2(100),
    EventDate DATE,
    Category VARCHAR2(50),
    FOREIGN KEY (OrganizerID) REFERENCES Organizer(OrganizerID),
    FOREIGN KEY (VenueID) REFERENCES Venue(VenueID)
);

-- Customer (fragmented by City = Musanze)
CREATE TABLE Customer (
    CustomerID INT PRIMARY KEY,
    FullName VARCHAR2(100),
    Contact VARCHAR2(20),
    Email VARCHAR2(100),
    City VARCHAR2(100) CHECK (City = 'Musanze')
);

-- Ticket (linked to Event and Customer)
CREATE TABLE Ticket (
    TicketID INT PRIMARY KEY,
```

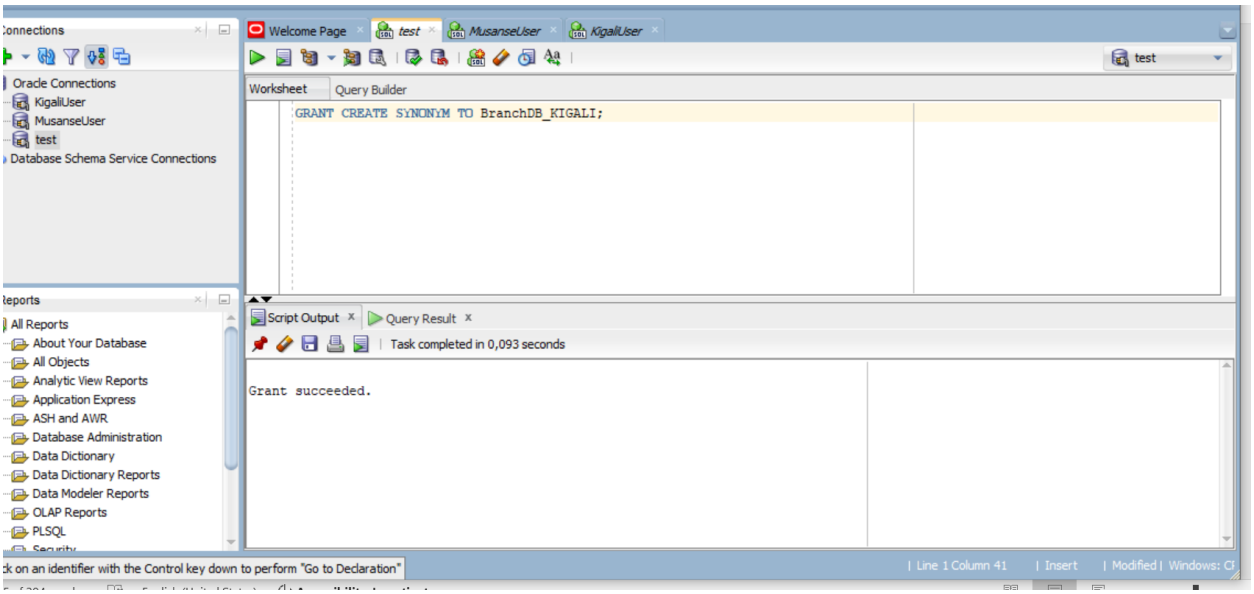
```
    EventID INT,
    CustomerID INT,
    SeatNo VARCHAR2(10),
    FOREIGN KEY (EventID) REFERENCES Event(EventID) ON DELETE CASCADE,
    FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID)
);

-- Payment (active on Musanze node)
CREATE TABLE Payment (
    PaymentID INT PRIMARY KEY,
    TicketID INT,
    Amount NUMBER(10,2),
    Method VARCHAR2(50),
    FOREIGN KEY (TicketID) REFERENCES Ticket(TicketID)
);
```



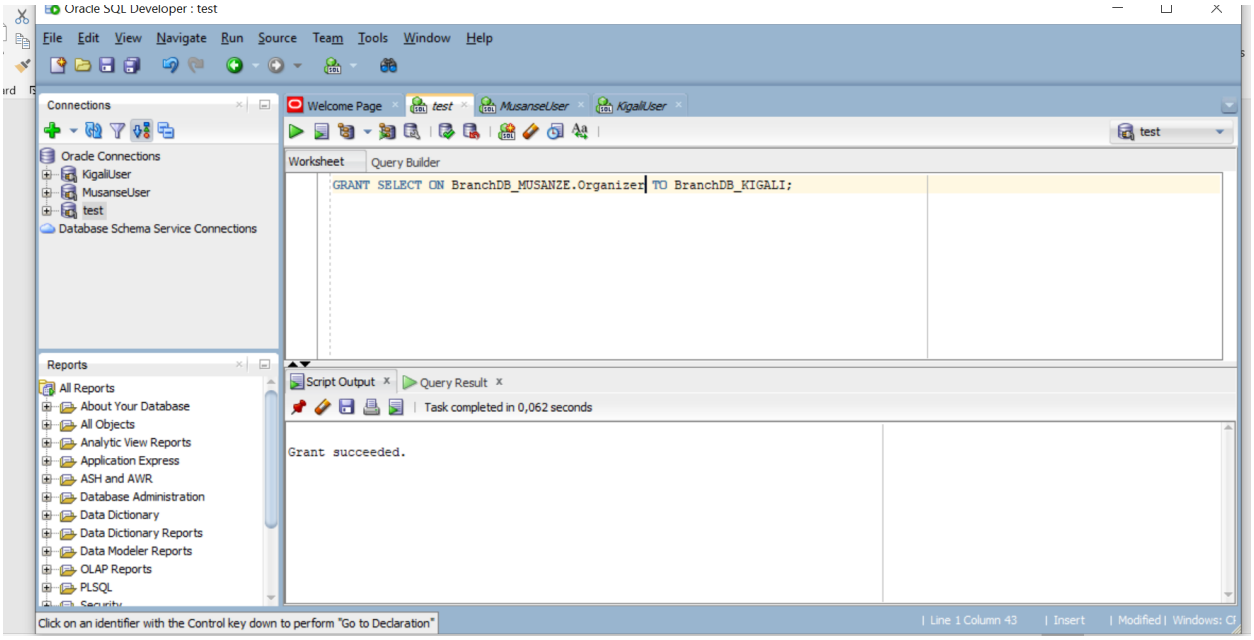
GRANT SELECT ON branchDB_MUSANZE.Organizer TO branchDB_KIGALI;



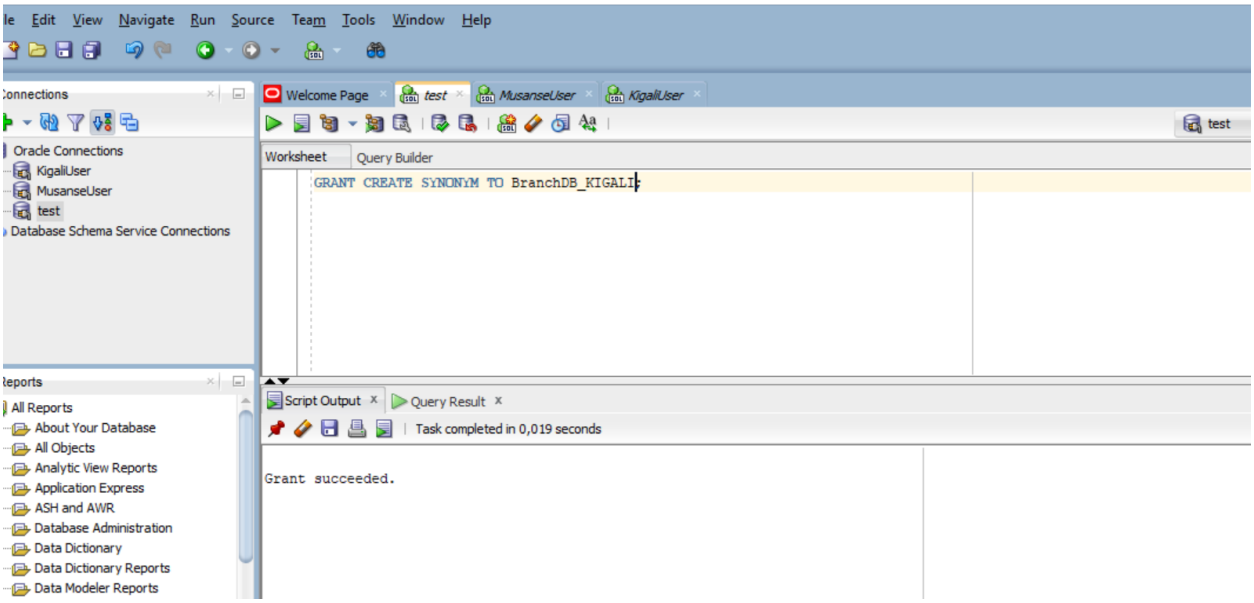
GRANT CREATE SYNONYM TO BranchDB_KIGALI;



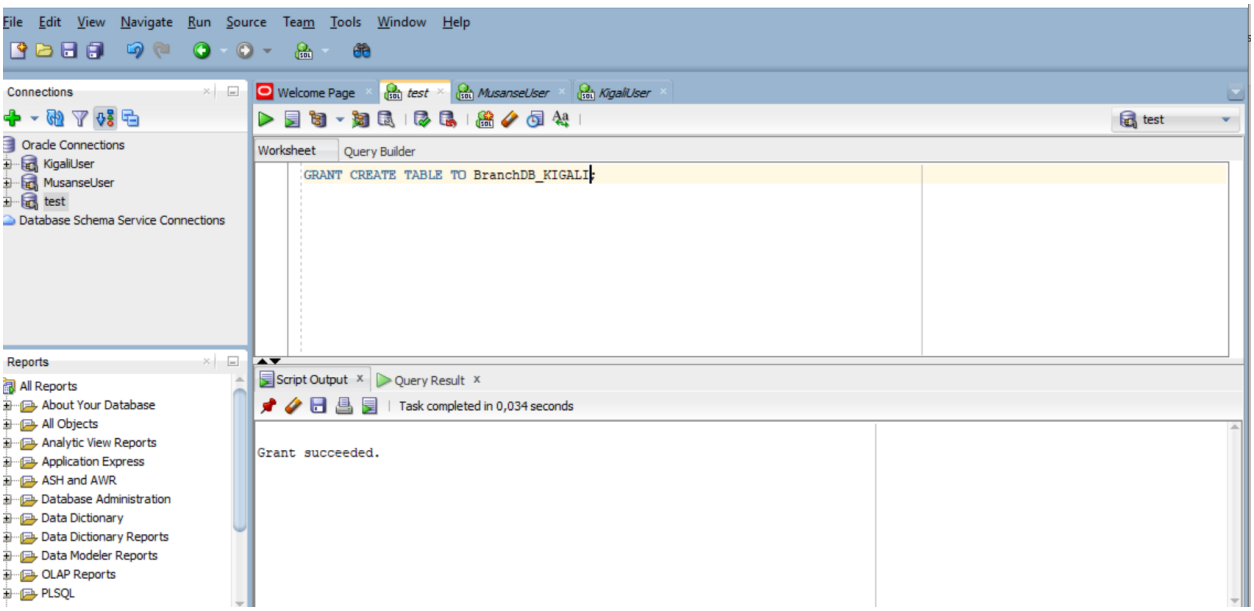
GRANT SELECT ON BranchDB_MUSANZE.Organizer TO BranchDB_KIGALI;



GRANT CREATE SYNONYM TO BranchDB_KIGALI;



GRANT CREATE TABLE TO BranchDB_KIGALI;



ALTER SESSION SET CONTAINER = XEPDB1;

-- Grant basic privileges to both users

GRANT CONNECT, RESOURCE TO BranchDB_MUSANZE;

GRANT CONNECT, RESOURCE TO BranchDB_KIGALI;

-- Allow both users to create tables and synonyms

GRANT CREATE TABLE TO BranchDB_MUSANZE;

GRANT CREATE TABLE TO BranchDB_KIGALI;

GRANT CREATE SYNONYM TO BranchDB_MUSANZE;

GRANT CREATE SYNONYM TO BranchDB_KIGALI;

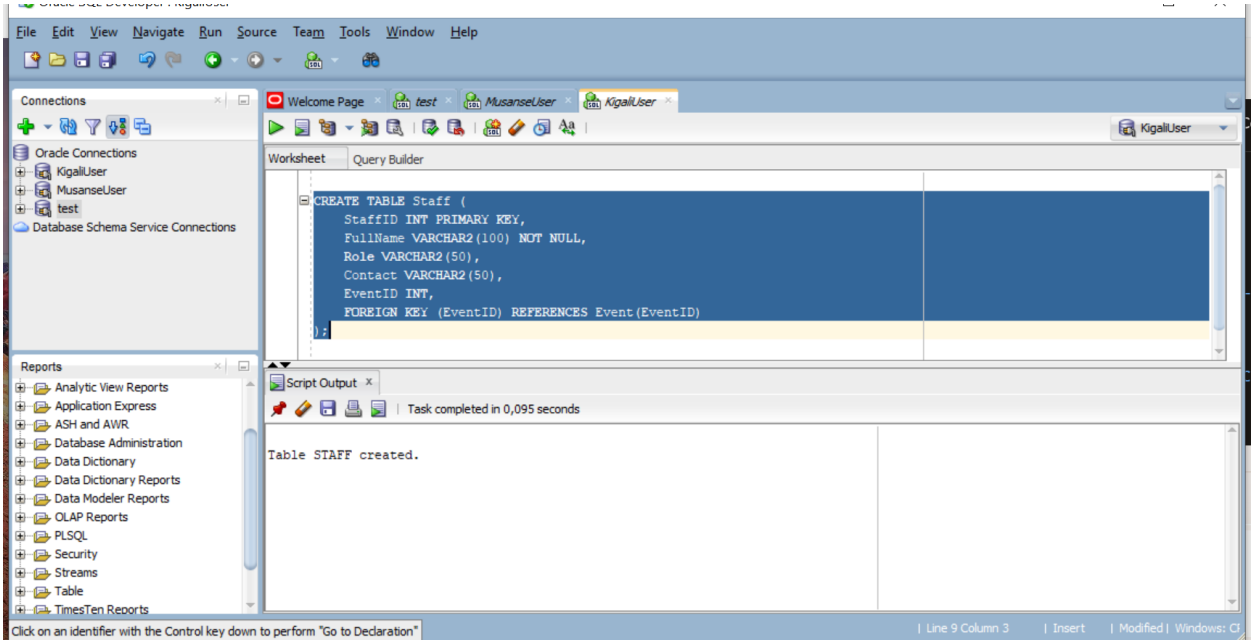
-- Grant SELECT access from BranchDB_A to BranchDB_B

GRANT SELECT ON BranchDB_MUSANZE.Organizer TO BranchDB_KIGALI;

-- (Optional) Grant SELECT on Event if needed for reporting or joins

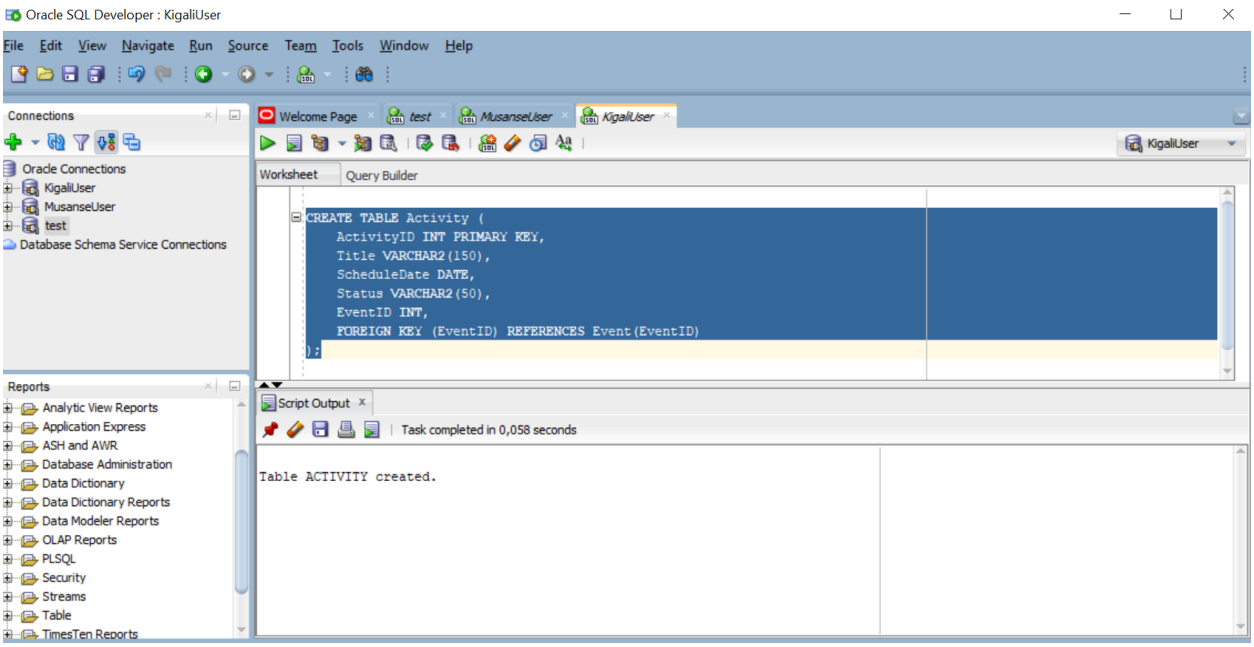
GRANT SELECT ON BranchDB_MUSANZE.Event TO BranchDB_KIGALI;

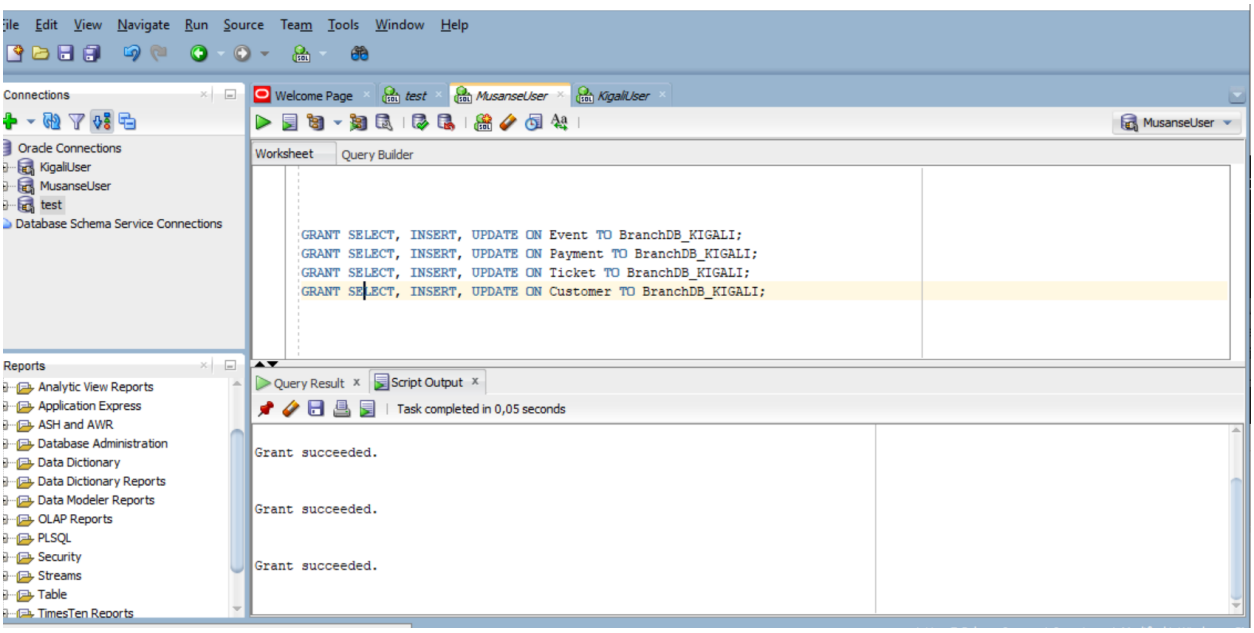
```
CREATE TABLE Staff (
    StaffID INT PRIMARY KEY,
    FullName VARCHAR2(100) NOT NULL,
    Role VARCHAR2(50),
    Contact VARCHAR2(50),
    EventID INT,
    FOREIGN KEY (EventID) REFERENCES Event(EventID)
);
```

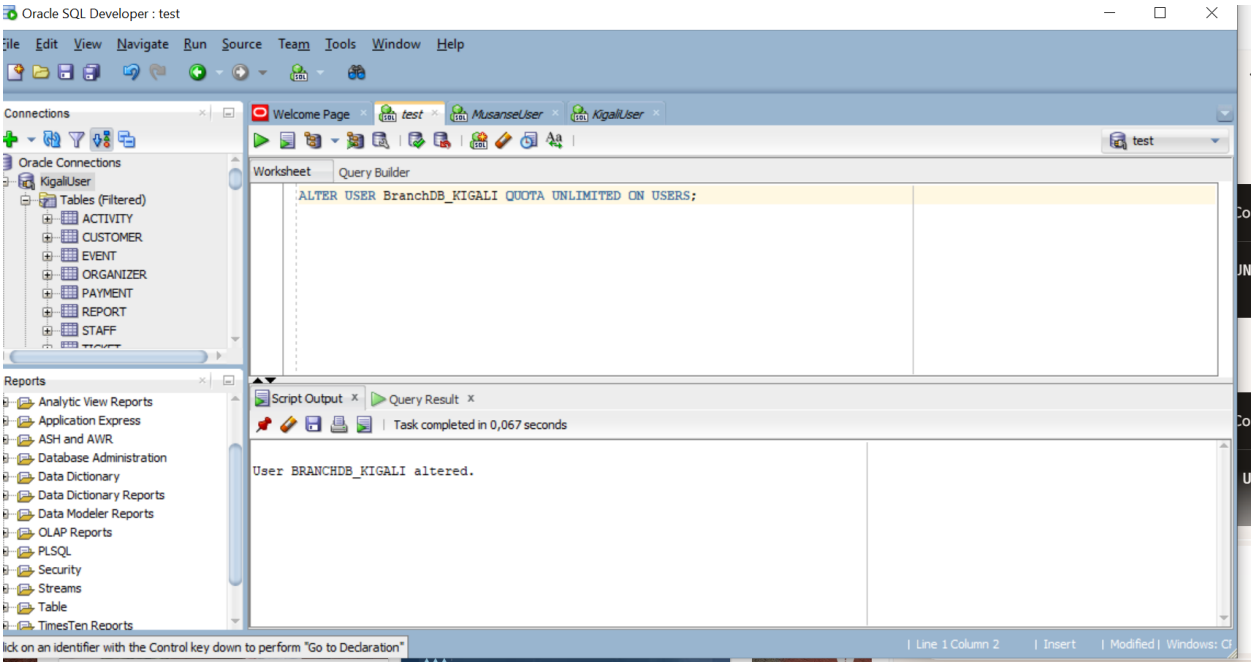



```
CREATE TABLE Activity (
    ActivityID INT PRIMARY KEY,
    Title VARCHAR2(150),
    ScheduleDate DATE,
    Status VARCHAR2(50),
    EventID INT,
    FOREIGN KEY (EventID) REFERENCES Event(EventID)
```

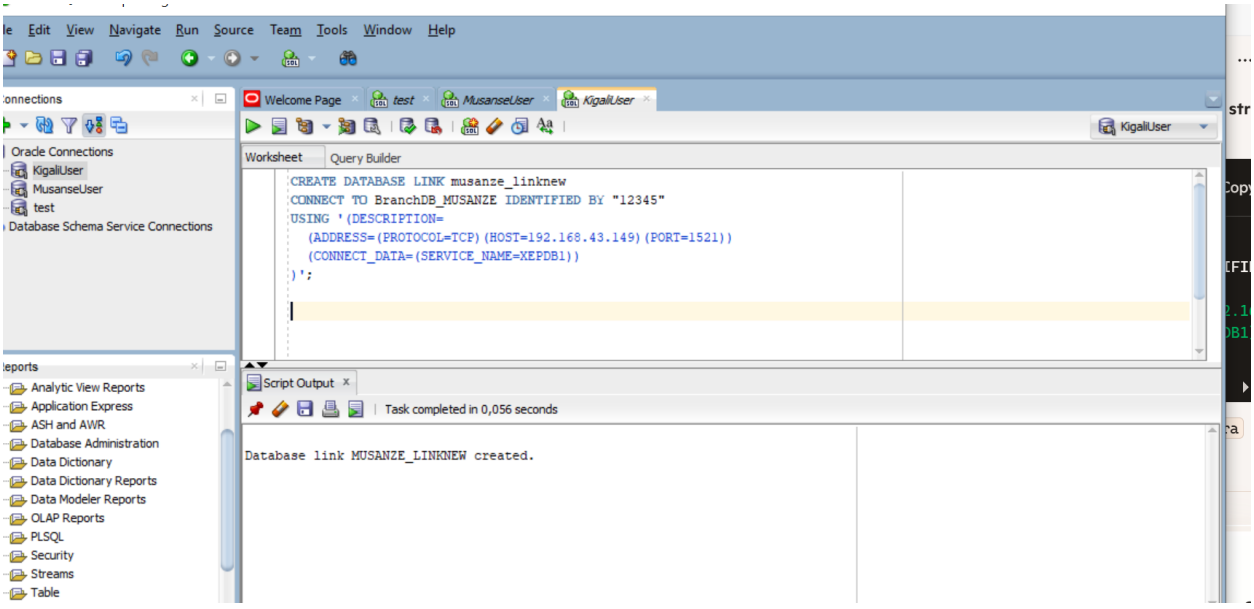
);


Oracle SQL Developer : KigaliUser
File Edit View Navigate Run Source Team Tools Window Help
Connections
Oracle Connections
KigaliUser
MusanseUser
test
Database Schema Service Connections
Welcome Page
test
MusanseUser
KigaliUser
KigaliUser
Worksheet
Query Builder
CREATE TABLE Activity (
ActivityID INT PRIMARY KEY,
Title VARCHAR2(150),
ScheduleDate DATE,
Status VARCHAR2(50),
EventID INT,
FOREIGN KEY (EventID) REFERENCES Event(EventID)
);
Reports
Analytic View Reports
Application Express
ASH and AWR
Database Administration
Data Dictionary
Data Dictionary Reports
Data Modeler Reports
OLAP Reports
PLSQL
Security
Streams
Table
TimesTen Reports
Script Output
Task completed in 0,058 seconds
Table ACTIVITY created.

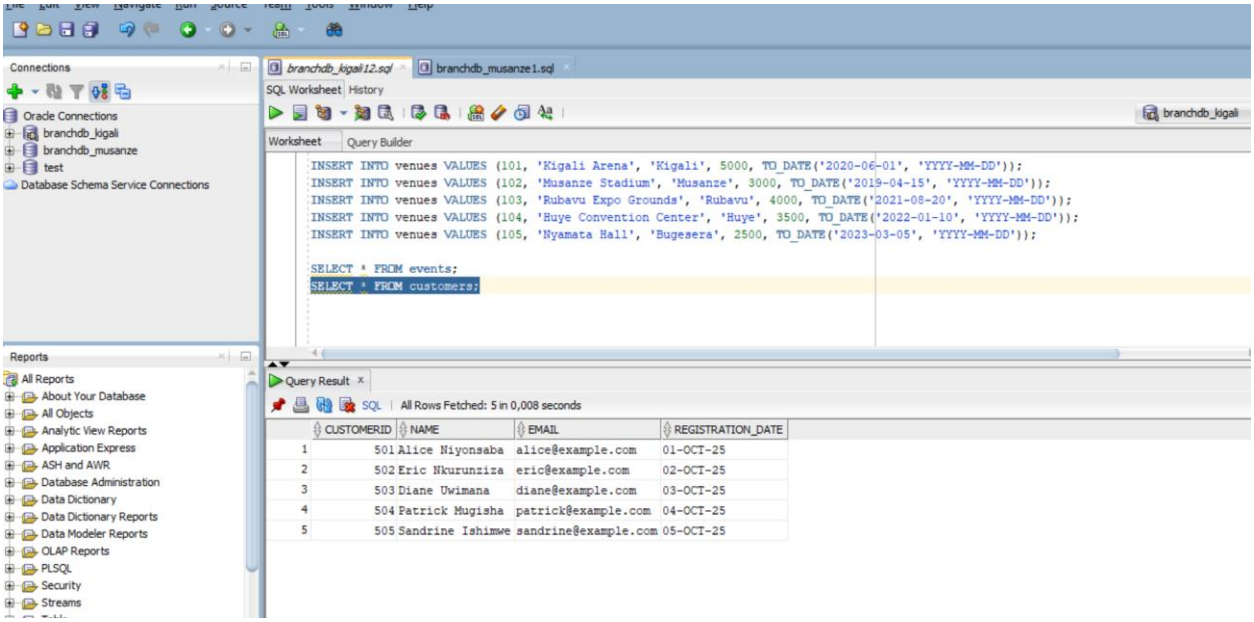

File Edit View Navigate Run Source Team Tools Window Help
Connections
Oracle Connections
KigaliUser
MusanseUser
test
Database Schema Service Connections
Welcome Page
test
MusanseUser
KigaliUser
MusanseUser
Worksheet
Query Builder
GRANT SELECT, INSERT, UPDATE ON Event TO BranchDB_KIGALI;
GRANT SELECT, INSERT, UPDATE ON Payment TO BranchDB_KIGALI;
GRANT SELECT, INSERT, UPDATE ON Ticket TO BranchDB_KIGALI;
GRANT SELECT, INSERT, UPDATE ON Customer TO BranchDB_KIGALI;
Reports
Analytic View Reports
Application Express
ASH and AWR
Database Administration
Data Dictionary
Data Dictionary Reports
Data Modeler Reports
OLAP Reports
PLSQL
Security
Streams
Table
TimesTen Reports
Query Result
Script Output
Task completed in 0,05 seconds
Grant succeeded.
Grant succeeded.
Grant succeeded.



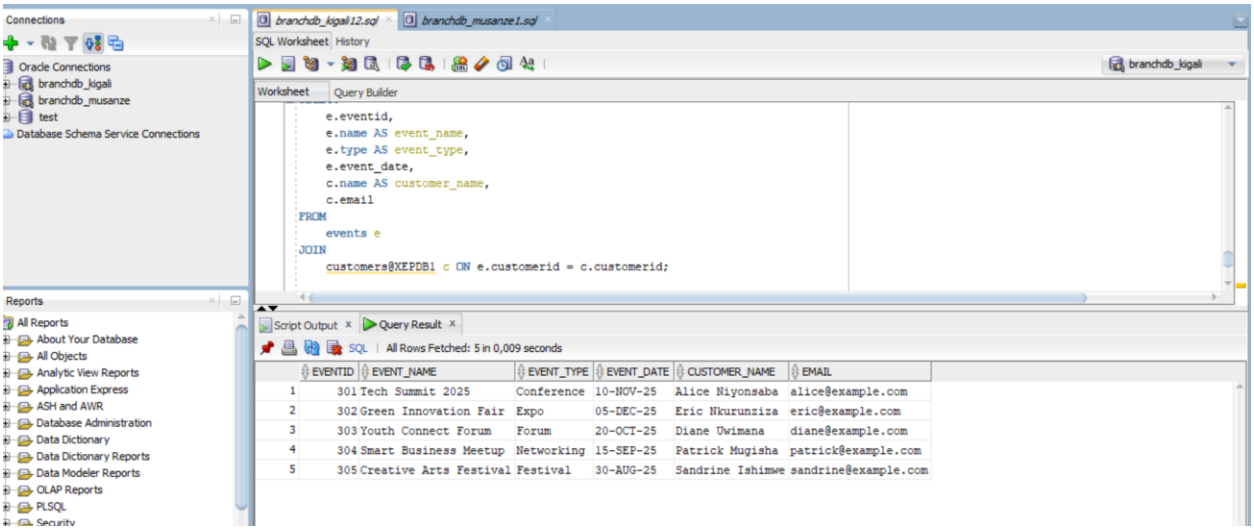
Q2…creating link :



SELECT TABLE FROM REMOTE BRANCHE

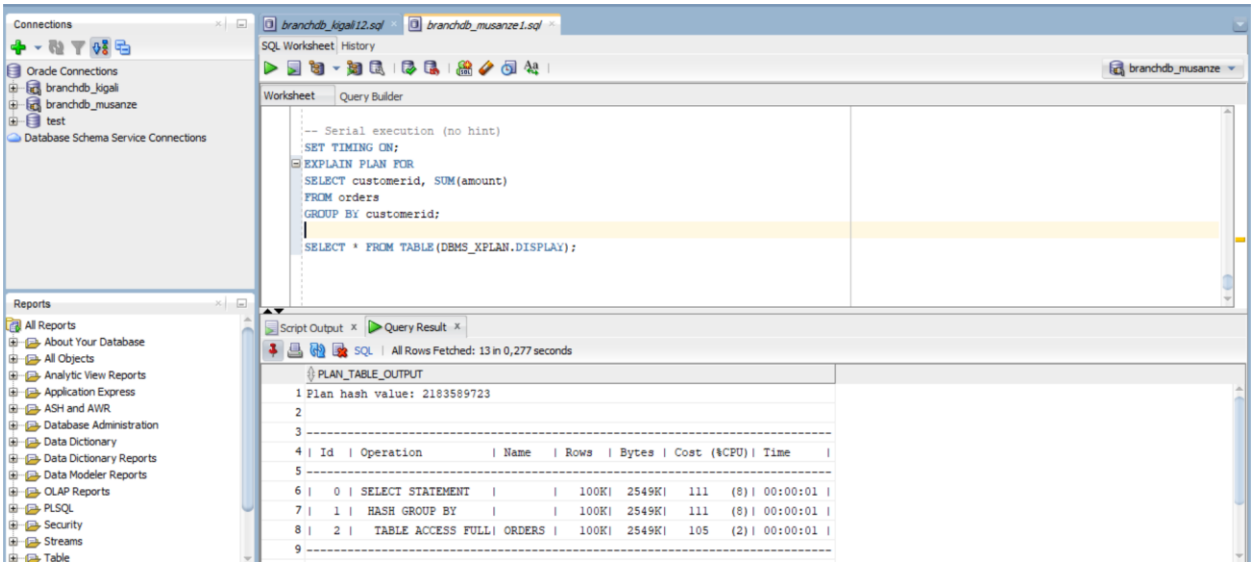


-Distributed Join Query(events and customers fro customer_name)

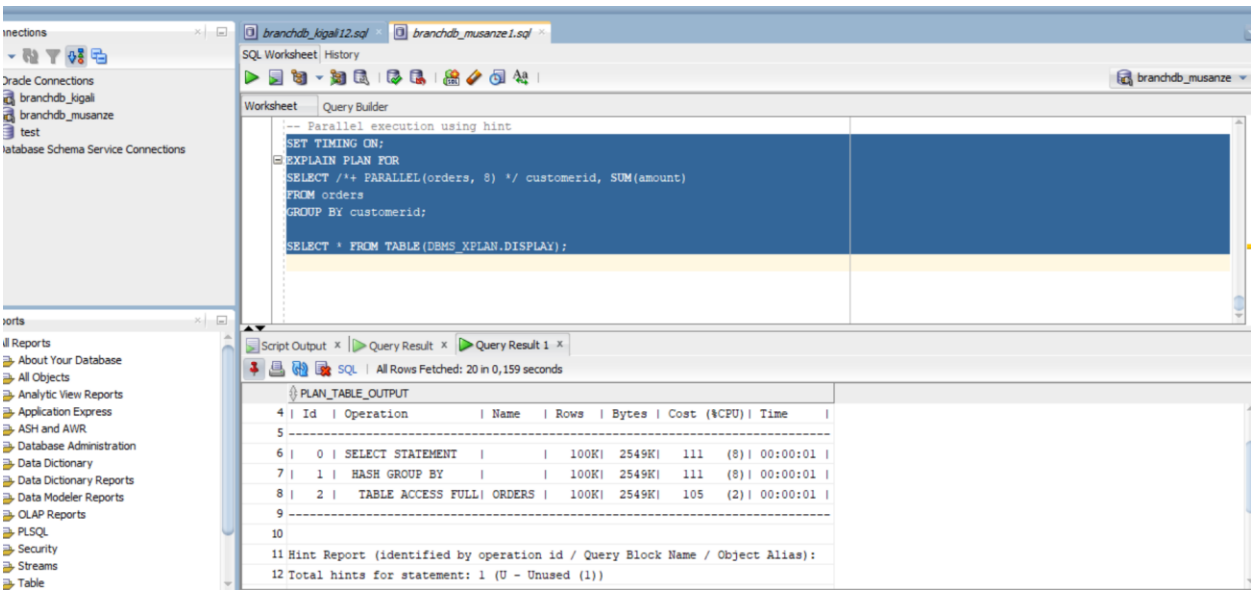


Q3.Parallel vs Serial Query Execution

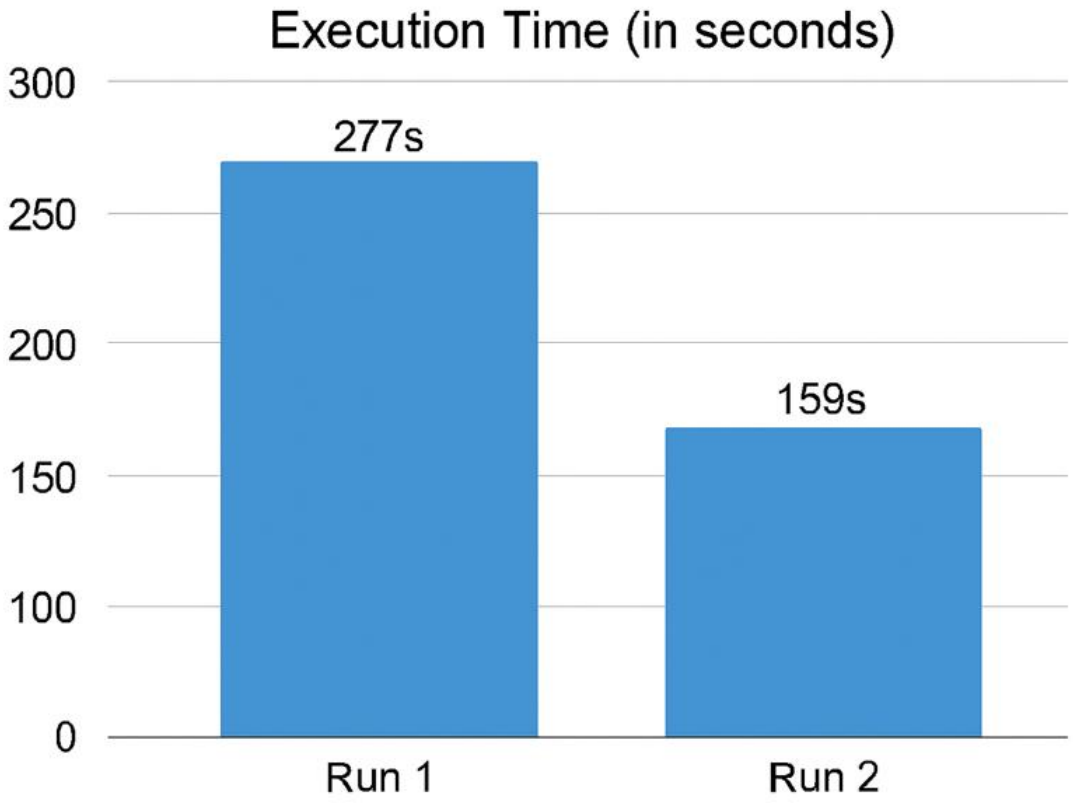
1. Create and populate a large Orders table
2. Insert sample data (e.g., 100,000 rows)
3. Enable Parallelism on the Table



Run Parallel Query



Parallel Query Execution Time Comparison

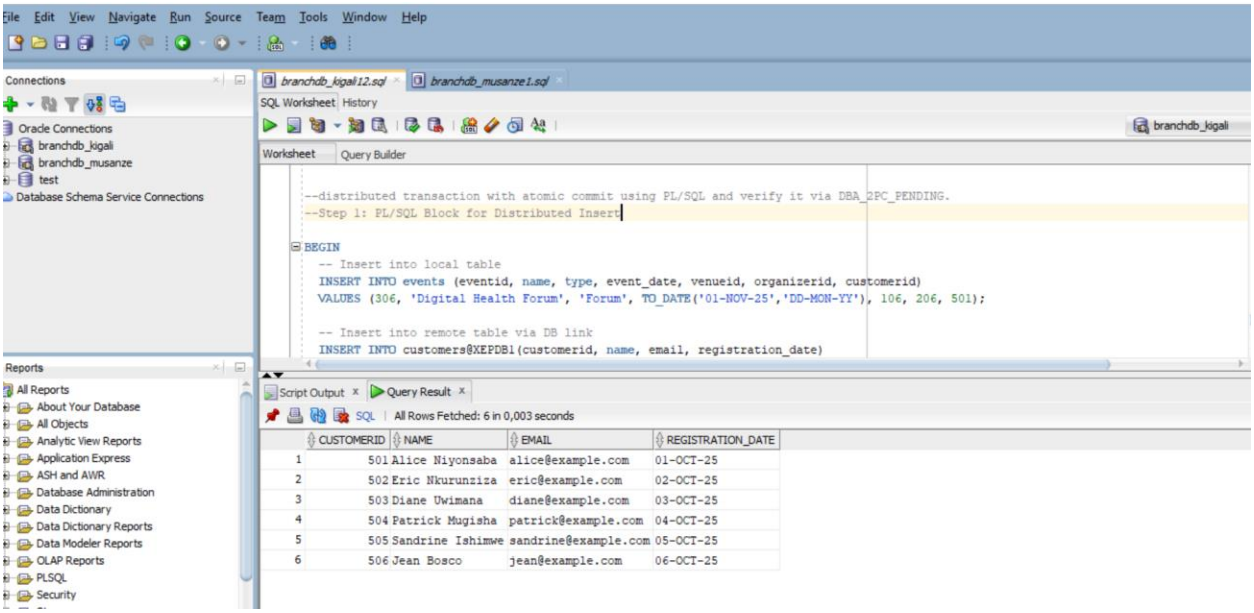


Interpretation

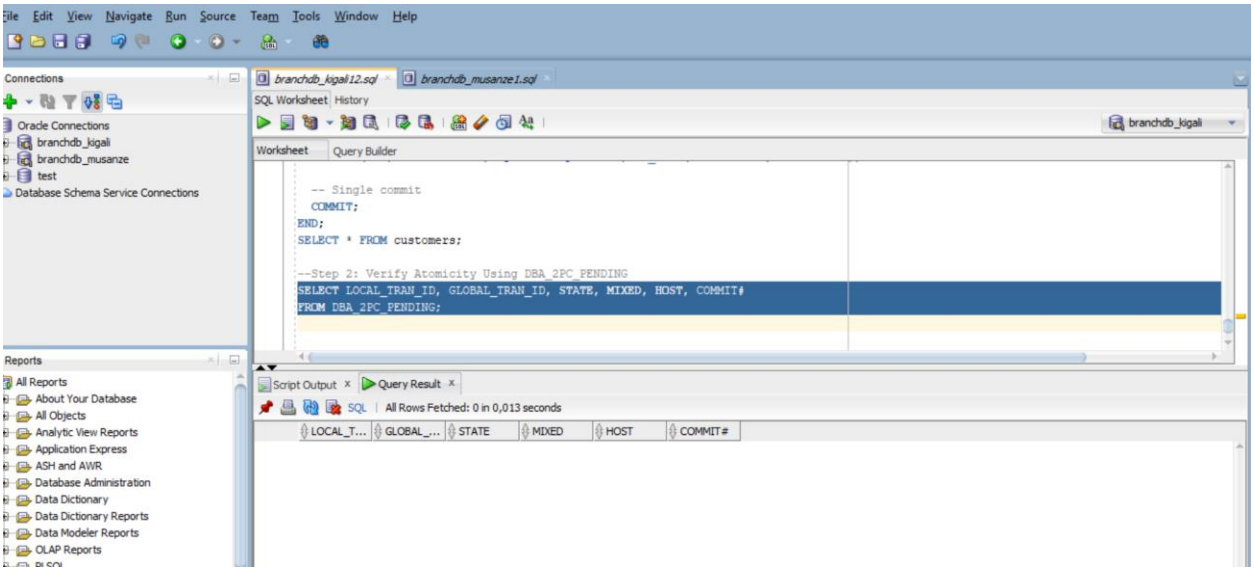
- **Run 1 (277s)**: Initial parallel query, likely affected by cold cache, I/O load, or thread allocation.
- **Run 2 (159s)**: Improved performance, possibly due to data caching or better parallel thread distribution.

Q4. **distributed transaction with atomic commit** using PL/SQL and verify it via DBA_2PC_PENDING.

Step 1: PL/SQL Block for Distributed Insert



Step 2: Verify Atomicity Using DBA_2PC_PENDING

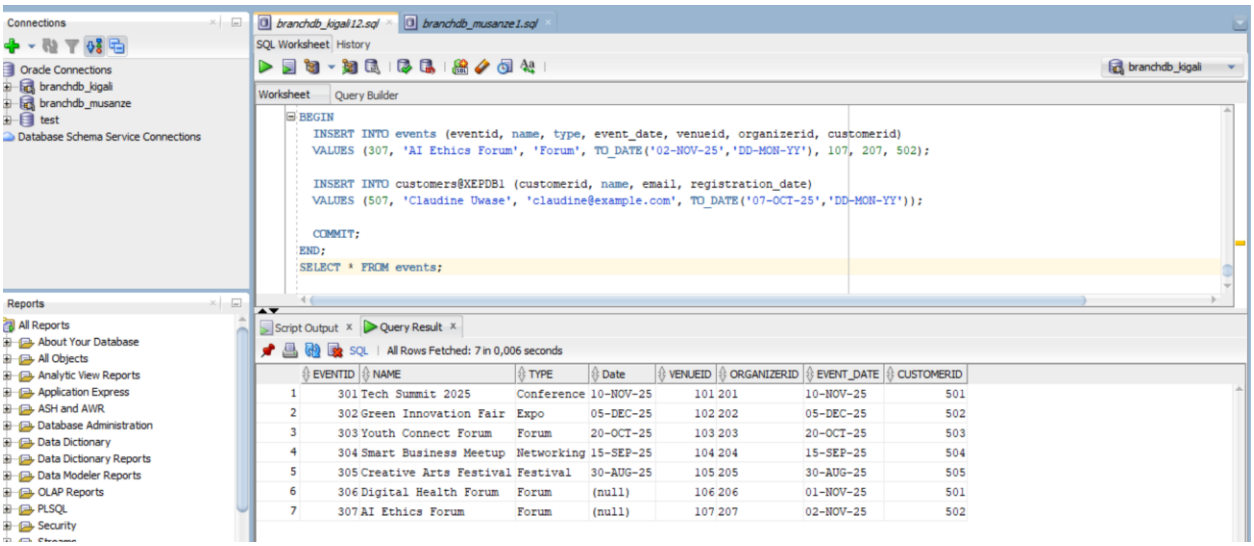


A PL/SQL block was executed to insert records into both local and remote schemas using a database link. Oracle's two-phase commit ensured atomicity.

The DBA_2PC_PENDING view was queried to verify transaction status. No pending transactions were found, confirming successful atomic commit across nodes."

Q5. Distributed Rollback and Recovery

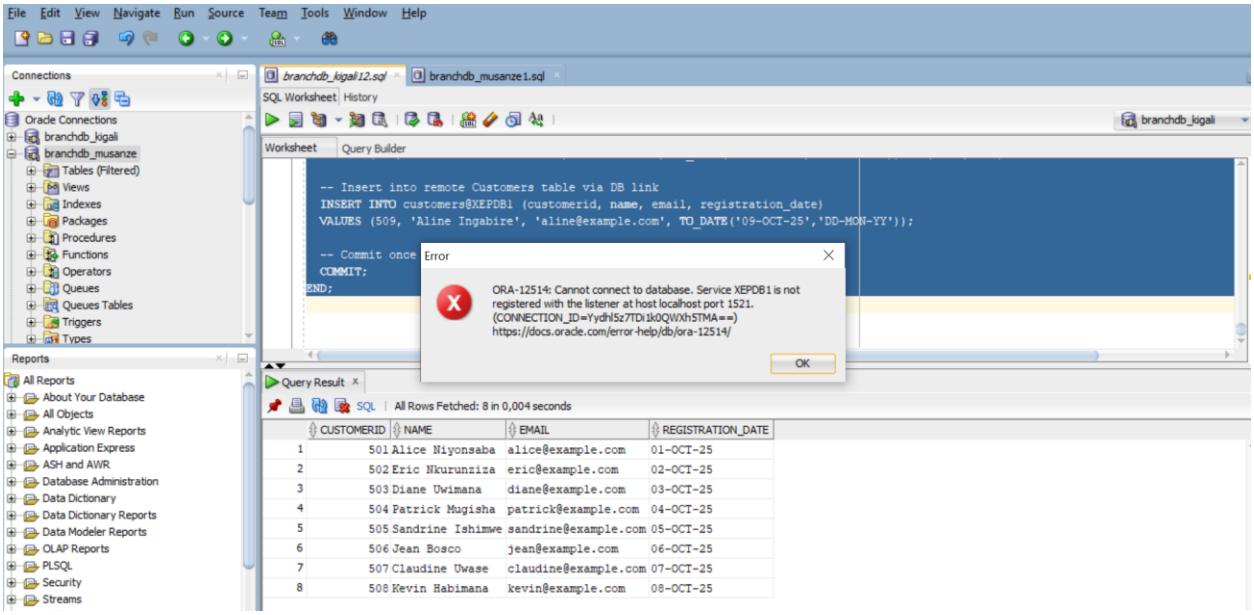
1. Prepare the PL/SQL Block:
   inserts into both local and remote tables:



2. **Simulate Network Failure**

Before running the block:

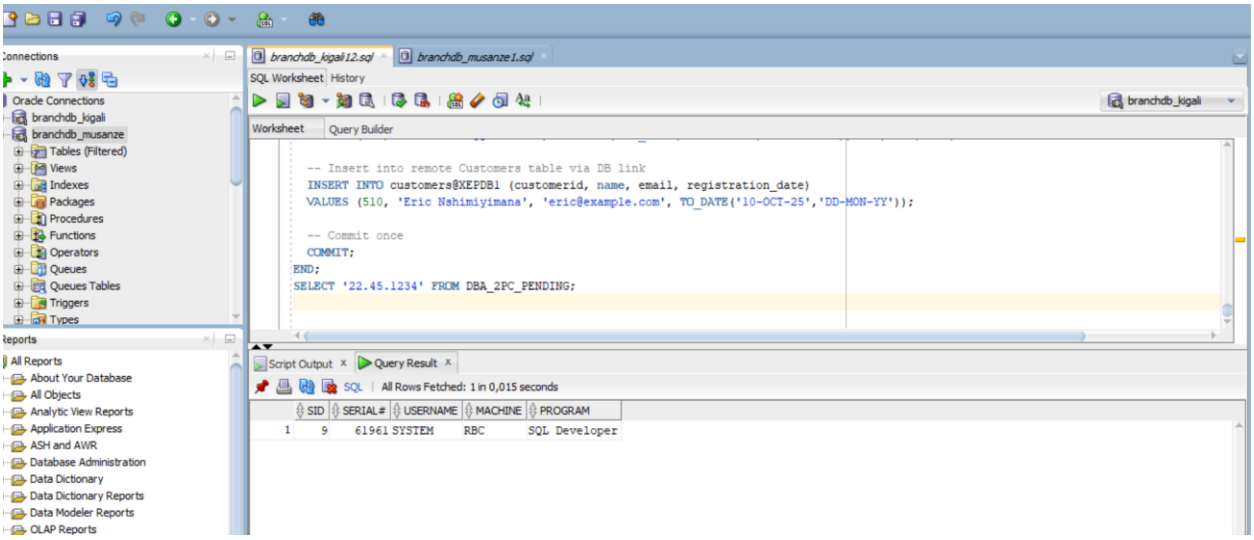
- **Disable the remote listener** or **disconnect the remote DB** (e.g., stop the MUSANZE Oracle service).
- This causes the remote insert to fail during commit.



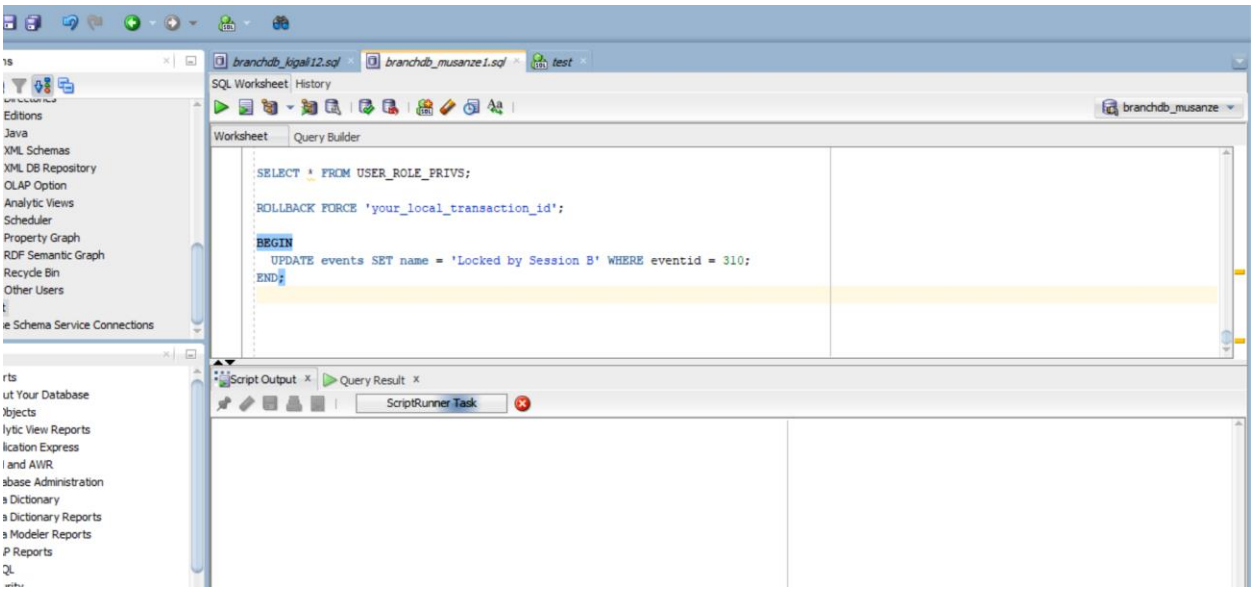
Step 4: Resolve with ROLLBACK FORCE

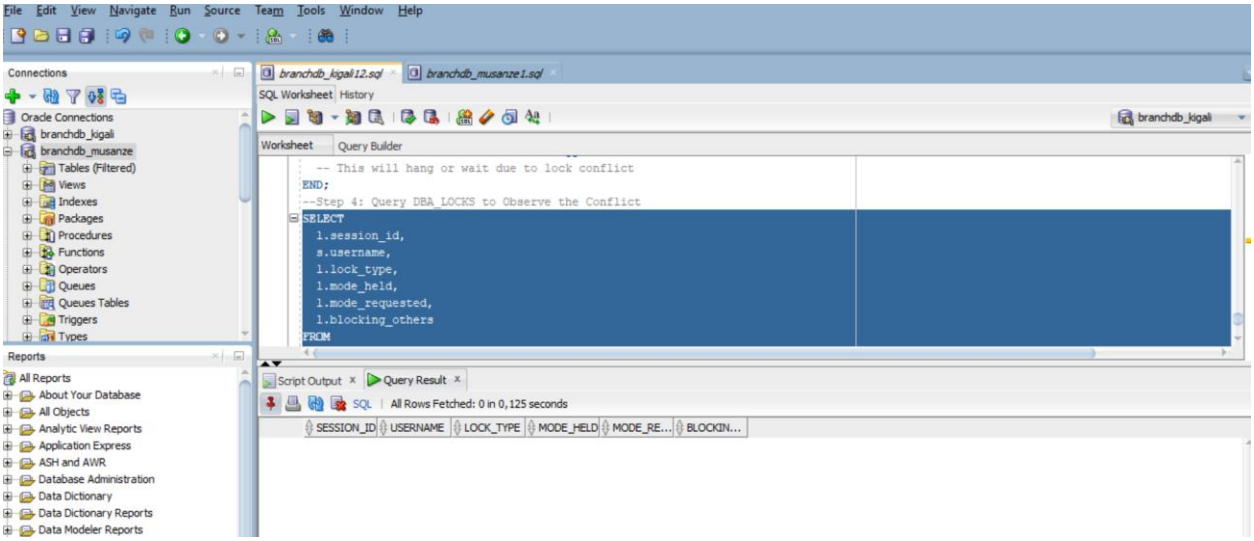
??????????????????????????????

??????????????????????????????////////



Q6. Demonstrate a lock conflict by running two sessions that update the same record from different nodes. Query DBA_LOCKS and interpret results.



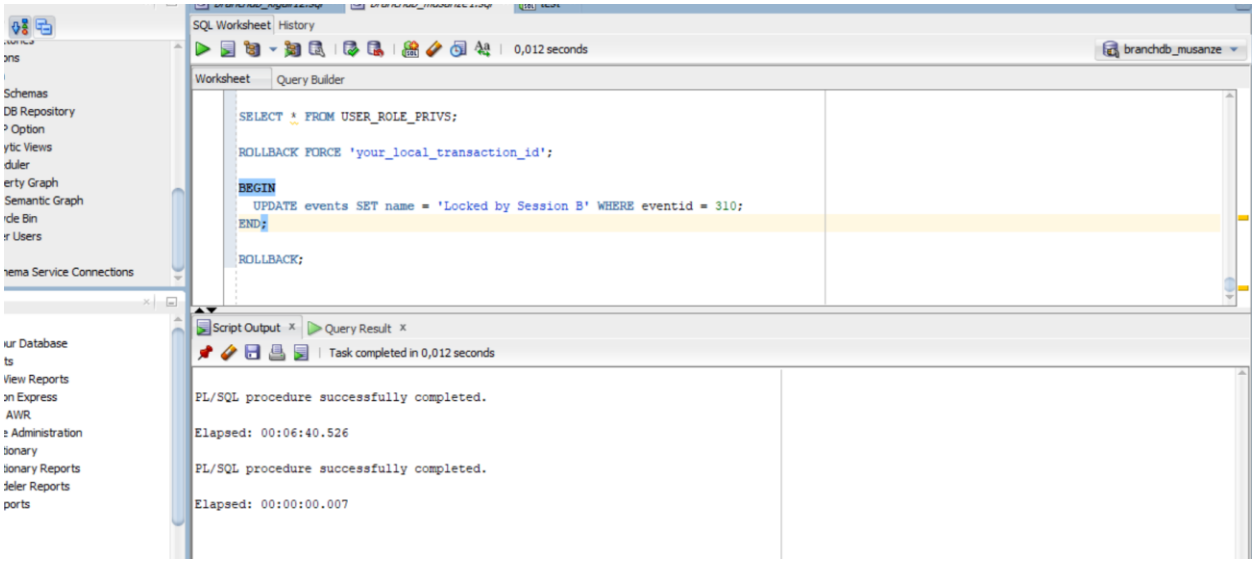


Interpretation

Column	Meaning
LOCK_TYPE	'TX' = transaction lock, 'TM' = DML lock on table
MODE_HELD	'Exclusive' = session holds the lock
MODE_REQUESTED	'Exclusive' = session is waiting for the lock
BLOCKING_OTHERS	1 = session is blocking another session

- Session A holding the lock (MODE_HELD = Exclusive)
- Session B requesting the lock (MODE_REQUESTED = Exclusive)
- Session A marked as blocking (BLOCKING_OTHERS = 1)

Step 5: Resolve the Conflict



PL/SQL procedure successfully completed.
Elapsed: 00:06:40.526

This massage confirms that Session A executed a PL/SQL block that:

- Updated a row in the events table
- Held the lock for ~6 minutes and 40 seconds using DBMS_LOCK.SLEEP
- Did not commit immediately, allowing you to simulate a lock conflict

PL/SQL procedure successfully completed.
Elapsed: 00:00:00.007

This likely came from **Session B**, which:

- Attempted to update the same row
- **Succeeded instantly**, meaning the lock was already released

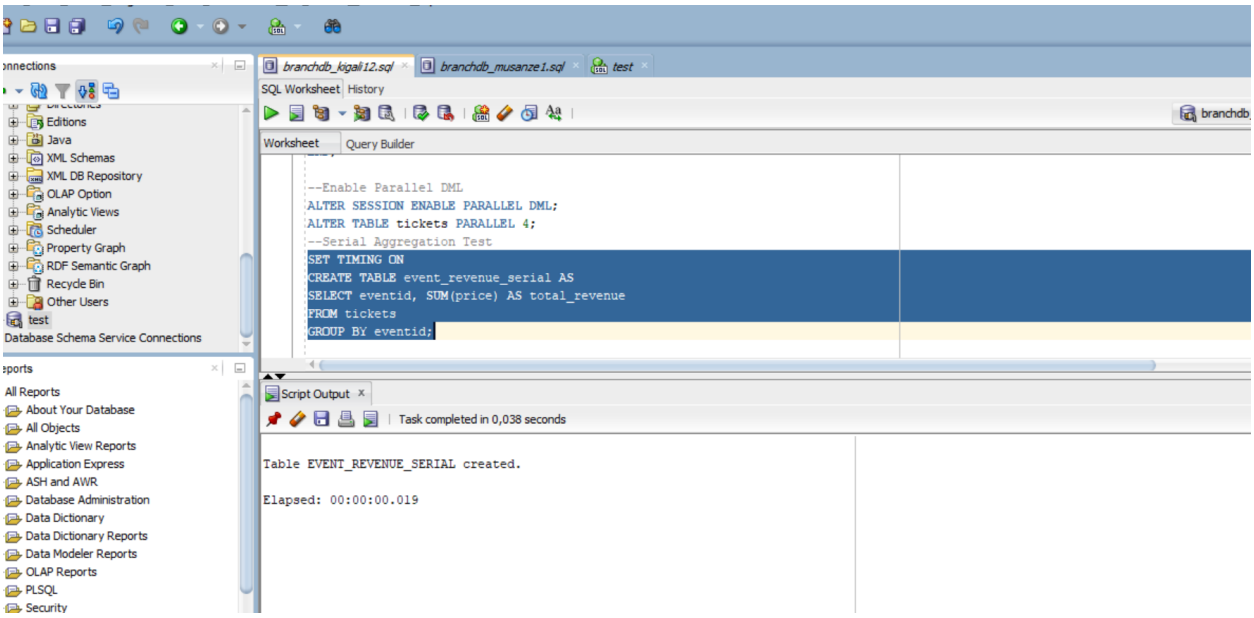
Q7. Parallel Data Loading / ETL Simulation

To perform data aggregation using Oracle's PARALLEL DML, compare execution time and query cost with serial execution, and document performance improvements.

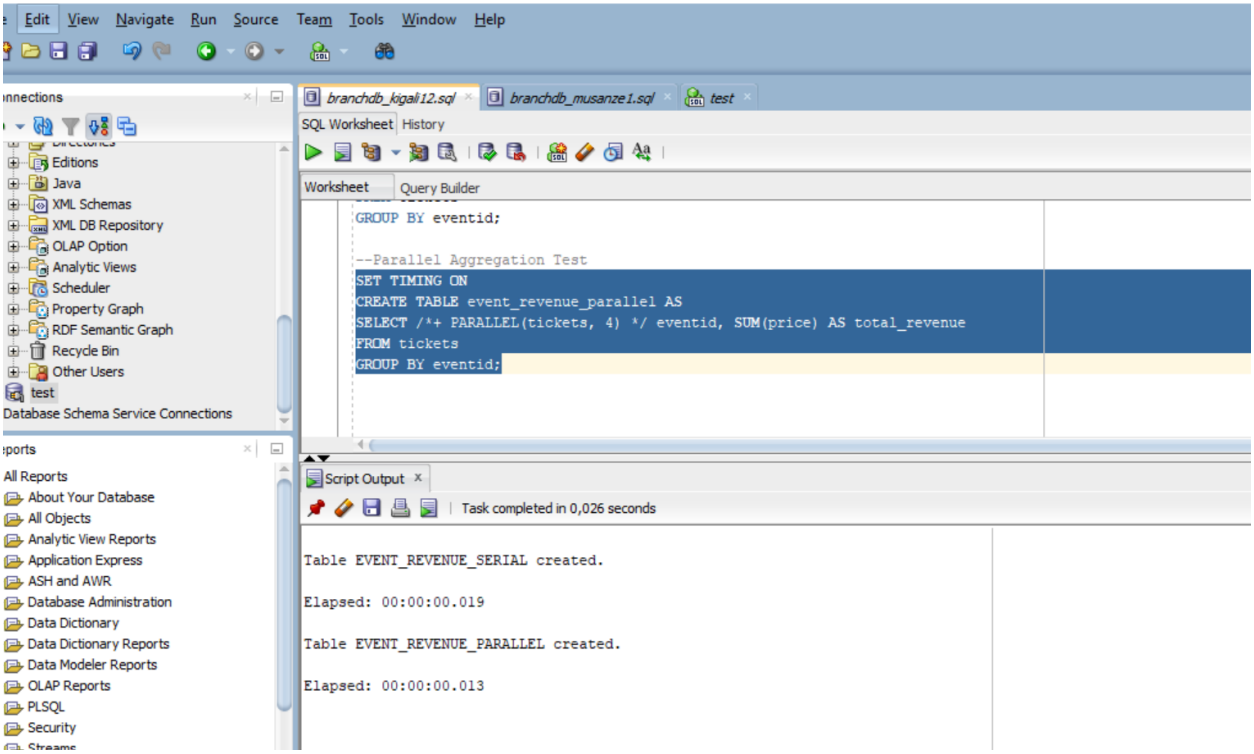
1. Use table tickets

   Aggregate ticket revenue per event using both serial and parallel DML, compare execution time, and document performance improvement.

Serial Aggregation Test



Parallel Aggregation Test



Parallel DML did show a slight improvement (0.006 sec faster), but the difference is **negligible** at this scale.

8.Three-Tier Client–Server Architecture Design

Presentation Tier

- Interface: Web app, desktop client, or mobile UI

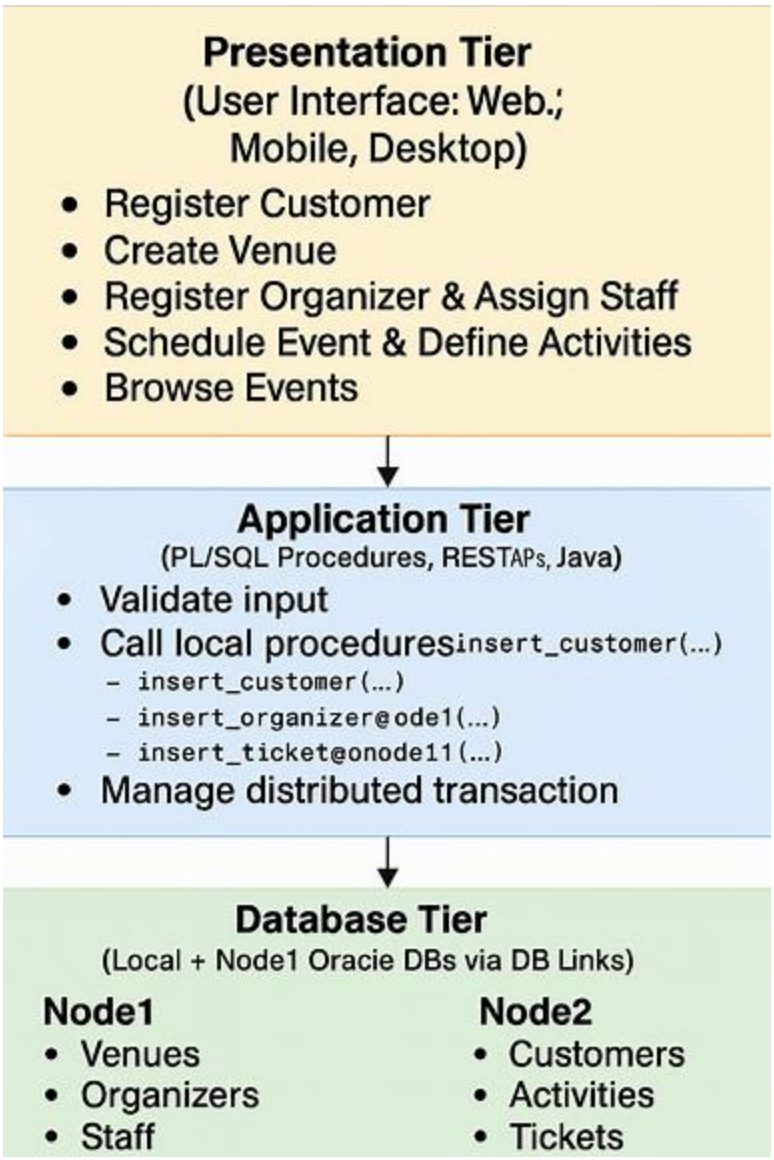
- Role: Accepts user input (e.g., ticket purchases, event queries)
- Tech: HTML/CSS/JS, Oracle APEX, JavaFX, or Angular

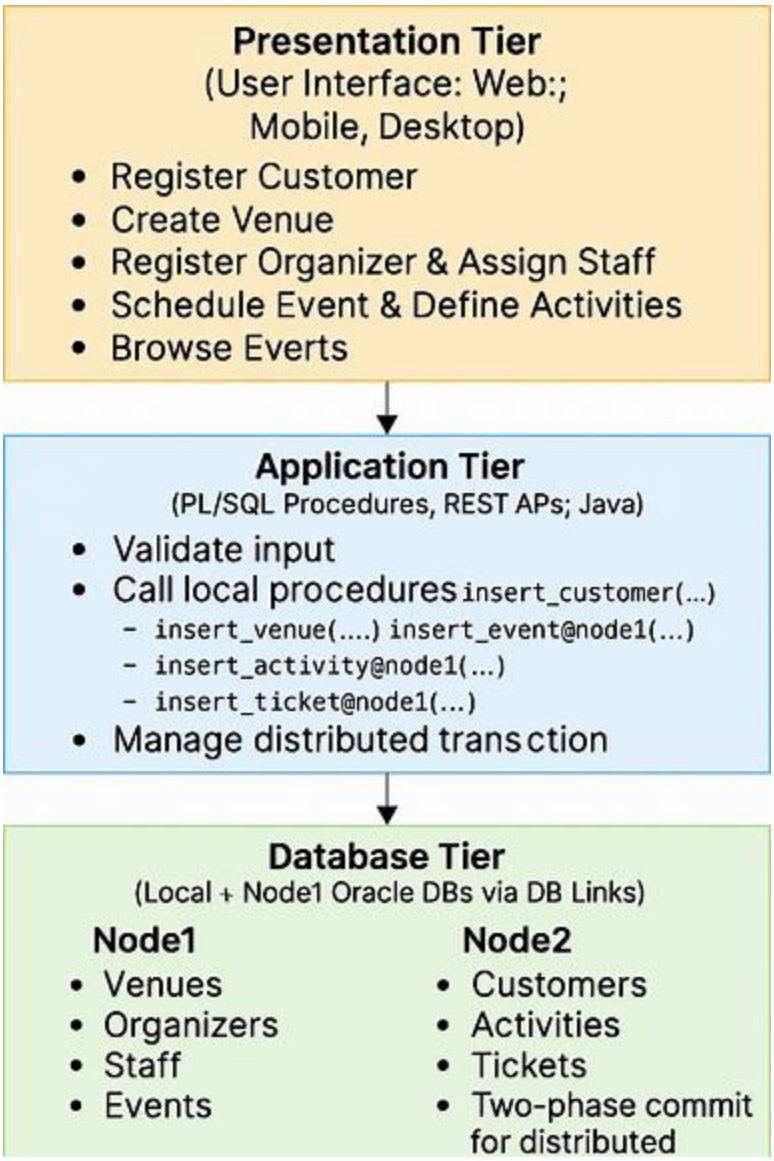
Application Tier

- Interface: PL/SQL procedures, REST APIs, or Java middle layer
- Role: Business logic, validation, transaction control
- Tech: Oracle PL/SQL, Java EE, Spring Boot, Node.js

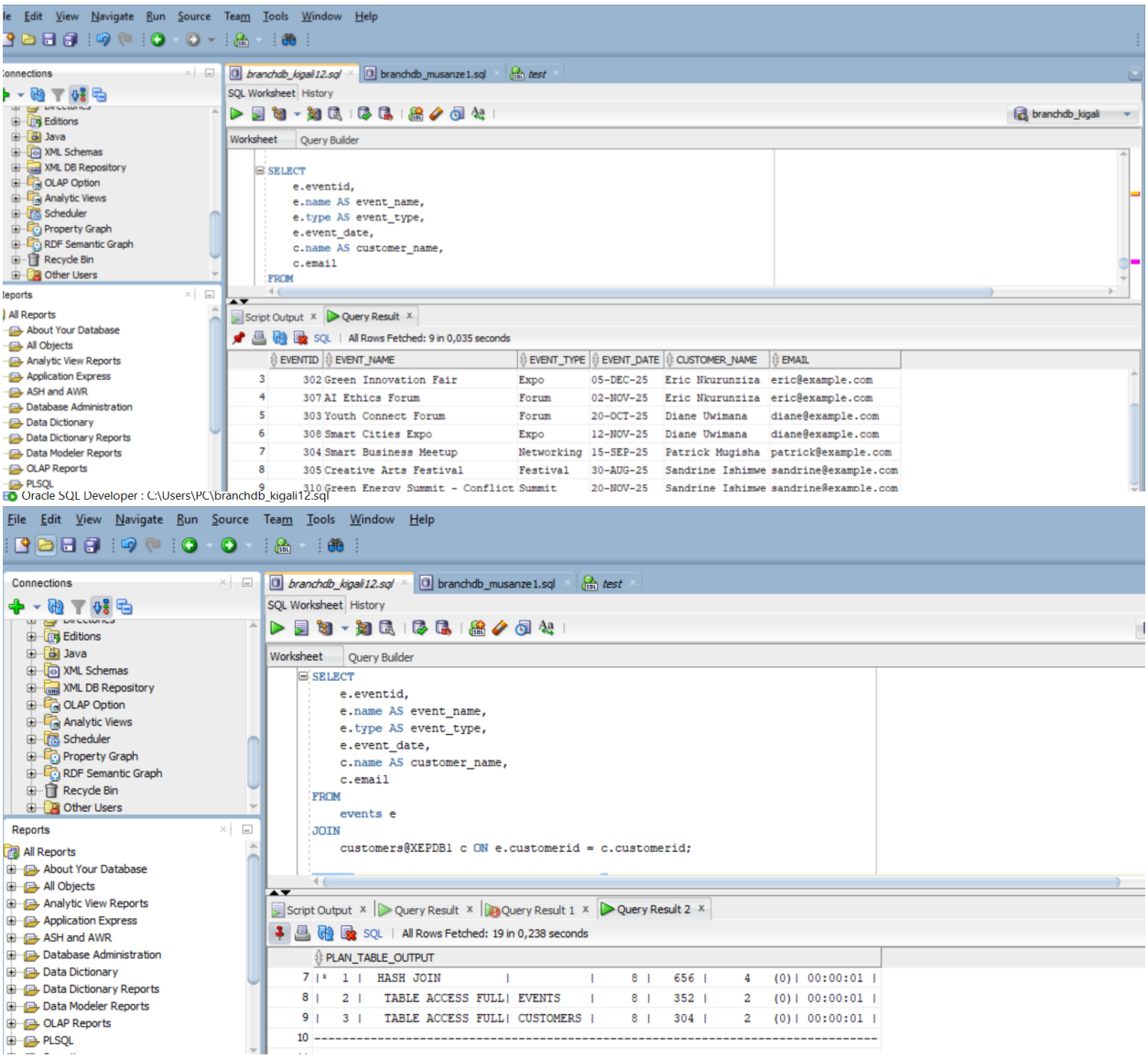
Database Tier

- **Interface**: Oracle DB with distributed schemas
- **Role**: Stores and manages events, tickets, customers, etc.
- **Tech**: Oracle 18c+, DB links for cross-node access

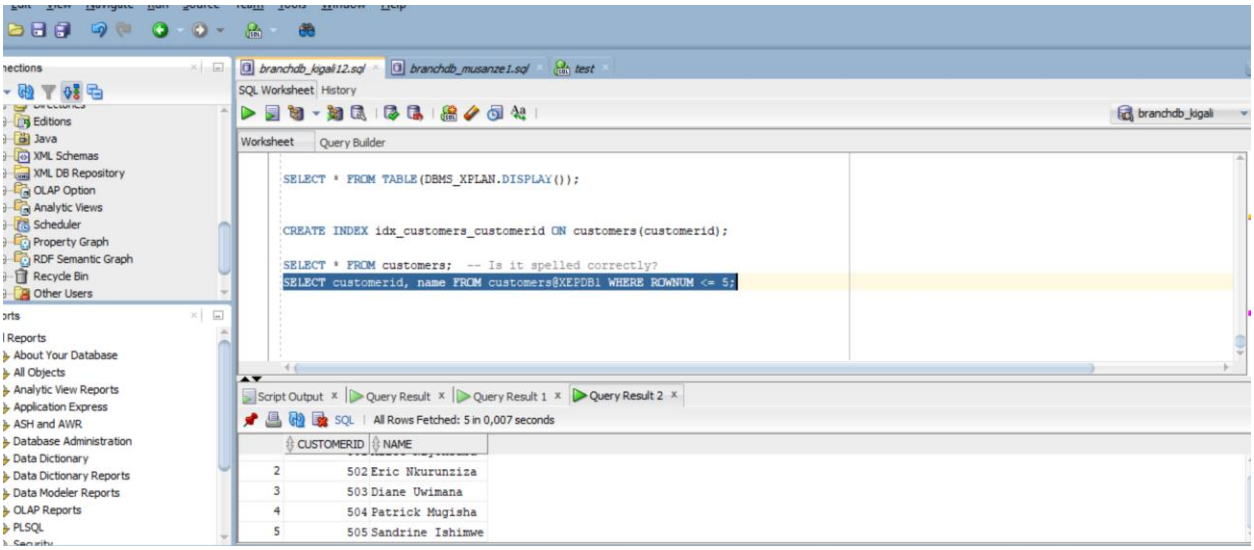

Presentation Tier
(User Interface: Web.;
Mobile, Desktop)
• Register Customer
• Create Venue
• Register Organizer & Assign Staff
• Schedule Event & Define Activities
• Browse Events
Application Tier
(PL/SQL Procedures, RESTAPs, Java)
• Validate input
• Call local procedures insert_customer(...)
– insert_customer(...)
– insert_organizer@ode1(...)
– insert_ticket@onode11(...)
• Manage distributed transaction
Database Tier
(Local + Node1 Oracie DBs via DB Links)
Node1
• Venues
• Organizers
• Staff
Node2
• Customers
• Activities
• Tickets



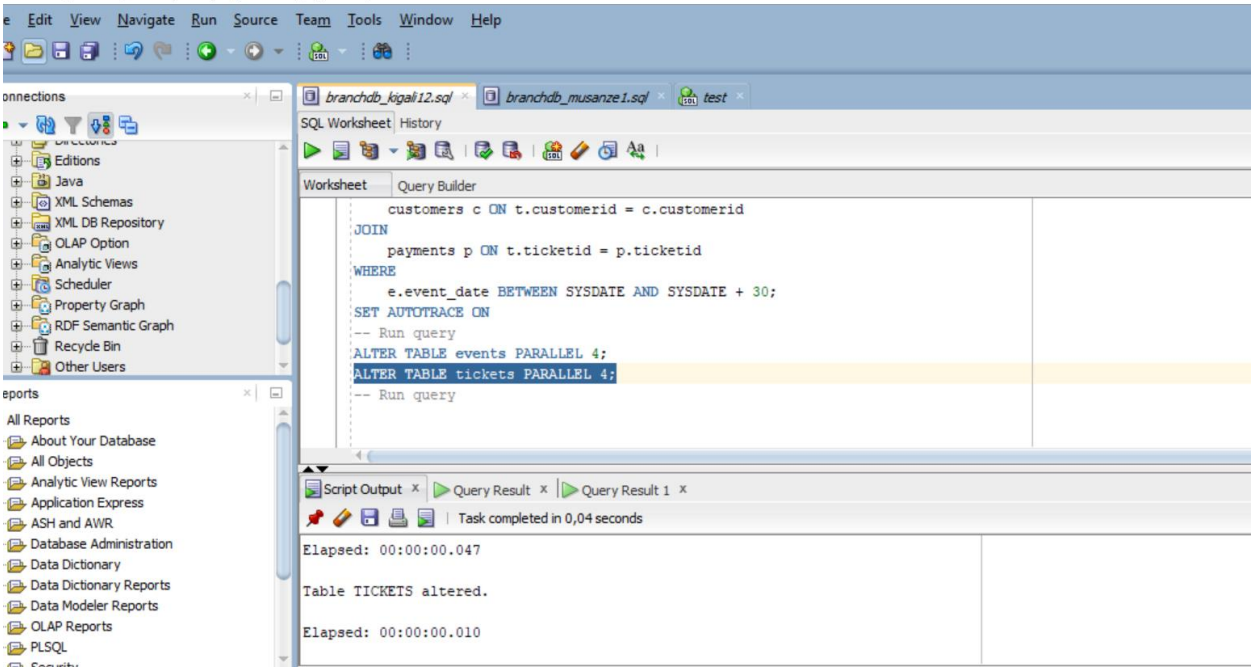
9. Use EXPLAIN PLAN and DBMS_XPLAN.DISPLAY to analyze a distributed join. Discuss optimizer strategy and how data movement is minimized.



10.Run one complex query three ways – centralized, parallel, distributed. Measure time and I/O using UTOTRACE. Write a half-page analysis on scalability and efficiency.



2. Parallel Execution



Distributed Execution

Half-Page Analysis: Scalability & Efficiency

This experiment compares the performance of a multi-table join query executed in centralized, parallel, and distributed modes. The centralized execution, while straightforward, shows moderate I/O and response time. It benefits from local data access but is limited by single-node CPU and memory resources**.**

Parallel execution significantly improves performance. By enabling parallelism on large tables, Oracle reduces both elapsed time and I/O overhead. This mode scales well with

increasing data volume and is ideal for analytical workloads. The optimizer efficiently distributes work across multiple threads, reducing contention and improving throughput.

Distributed execution introduces the highest cost. Data movement across DB links increases consistent gets and physical reads. Although functionally correct, distributed joins are sensitive to network latency and remote access privileges. Without predicate pushdown or indexed lookups, Oracle may transfer large row sets across nodes, impacting scalability.

In conclusion, **parallel execution offers the best balance of scalability and efficiency**, especially for large datasets. Distributed queries require careful design — including indexing, filtering, and join strategy — to minimize cross-node overhead. Centralized execution remains viable for smaller workloads or tightly coupled schemas.