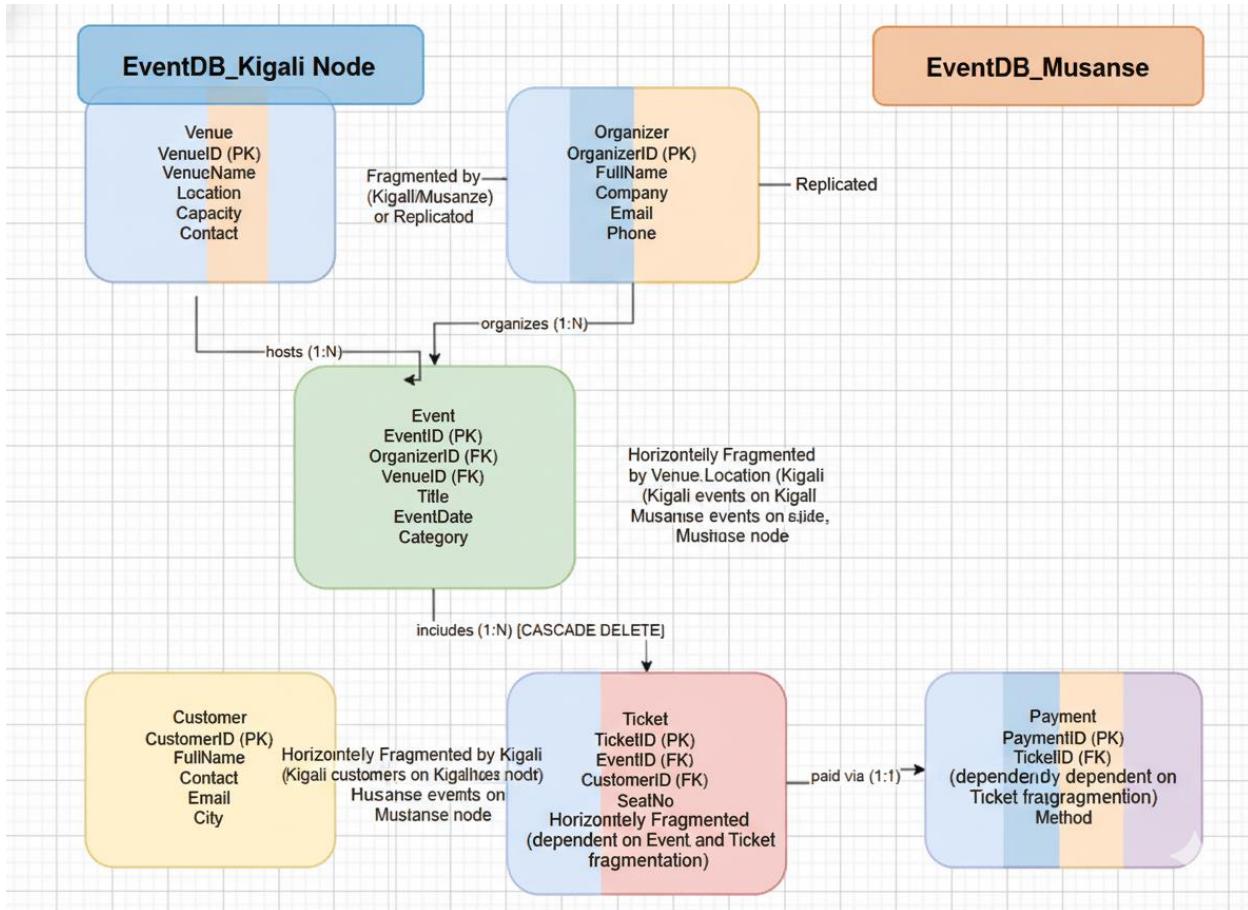


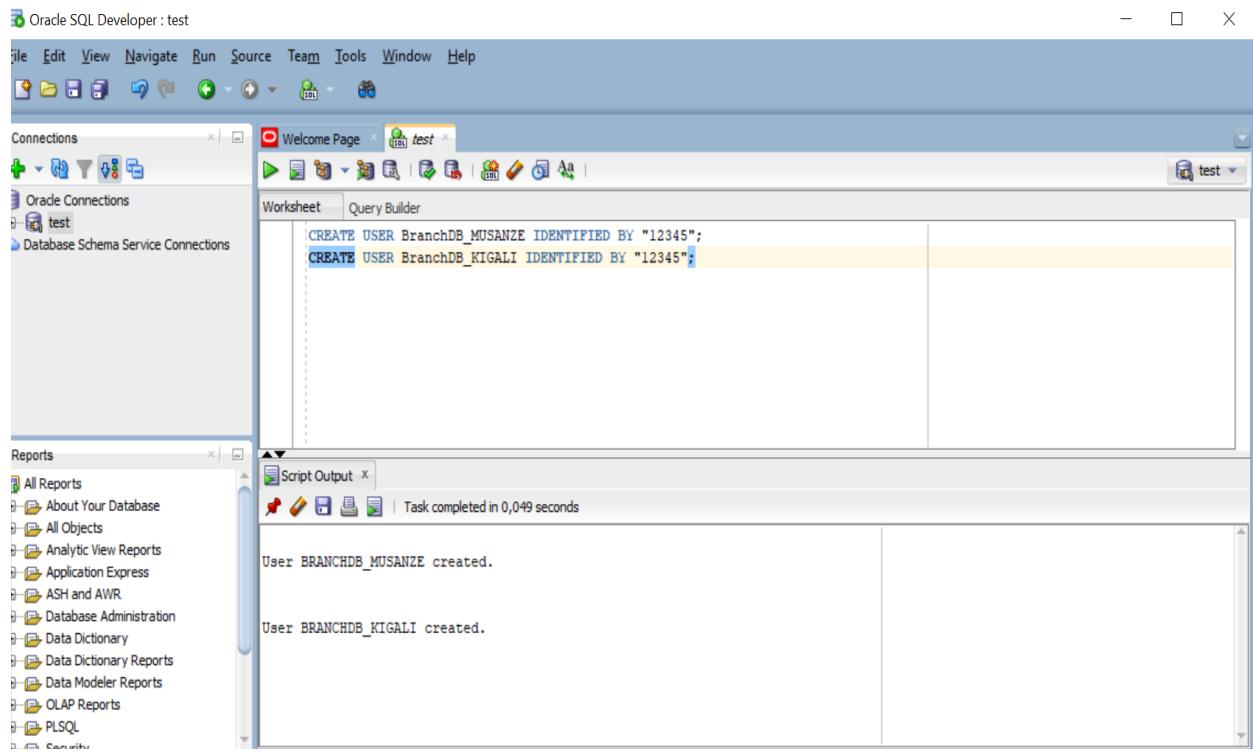
## Topic: Parallel and Distributed Databases

Entity Relationship Diagram (nod\_a and node\_b)



Q1..CREATE USER BranchDB\_MUSANZE IDENTIFIED BY "12345";

```
CREATE USER BranchDB_KIGALI IDENTIFIED BY "12345";
```



The screenshot shows the Oracle SQL Developer interface. In the top-left pane, there's a tree view of connections, with 'test' selected under 'Oracle Connections'. The main workspace contains a 'Worksheet' tab with the following SQL code:

```
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;
```

Below the worksheet, a 'Script Output' window displays the results of the executed commands:

```
Grant succeeded.  
Grant succeeded.
```

```
GRANT CONNECT, RESOURCE TO BranchDB_MUSANZE;
```

```
GRANT CONNECT, RESOURCE TO BranchDB_KIGALI;
```

## Creating tables

```
CREATE TABLE Organizer (
```

```
    OrganizerID INT PRIMARY KEY,
```

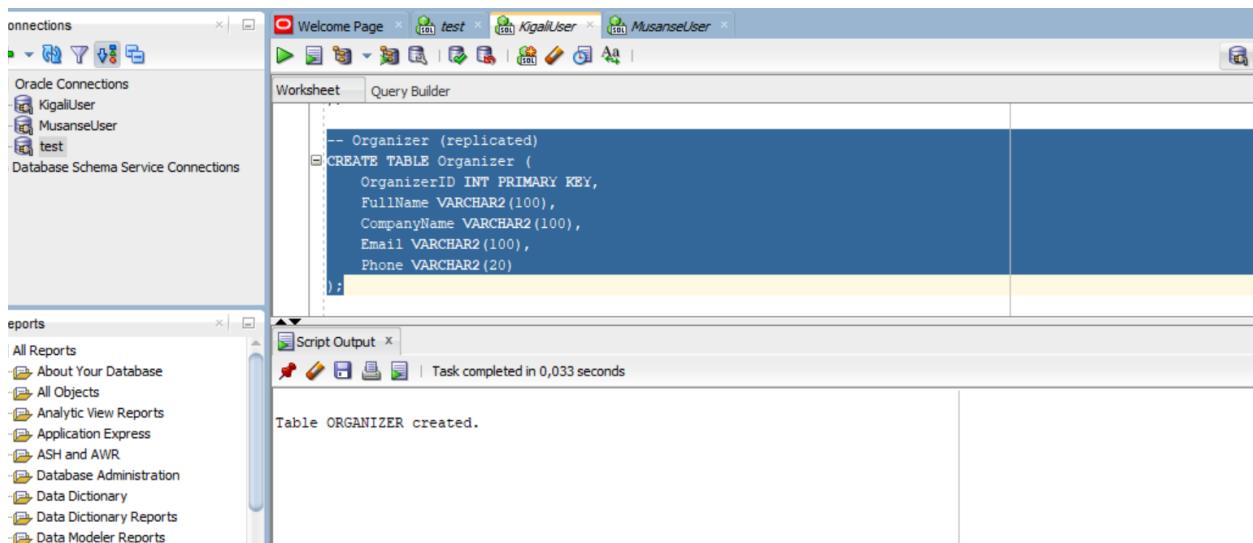
```
    FullName VARCHAR2(100),
```

```
    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)
);

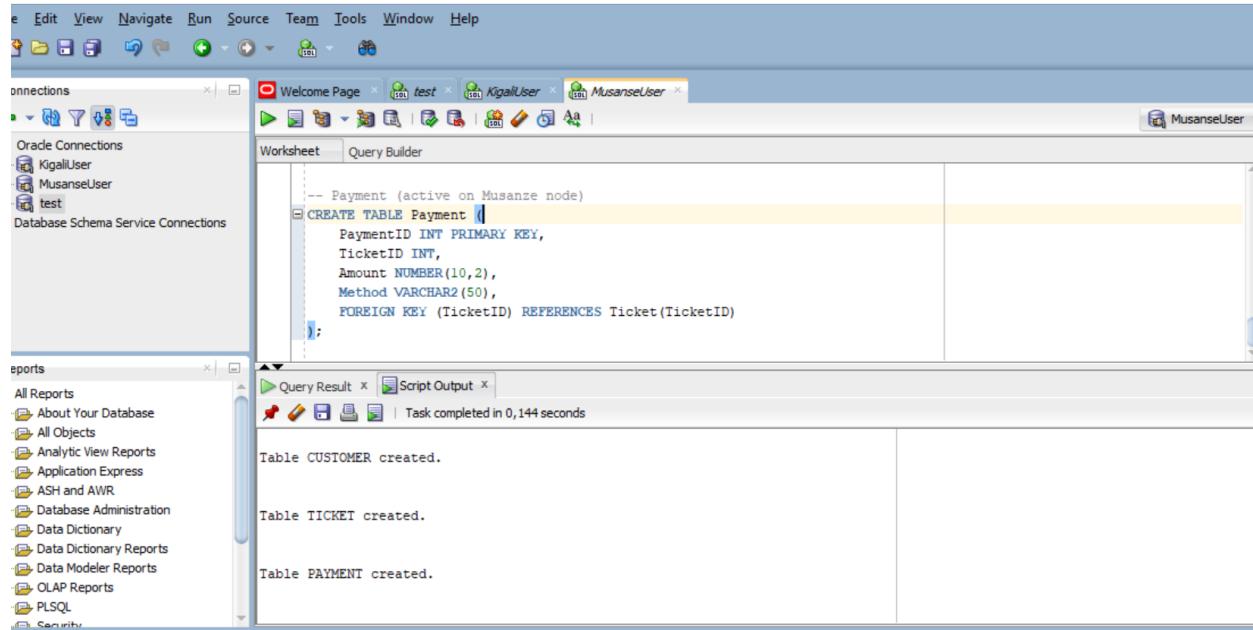
```

-- Payment (active on Musanze node)

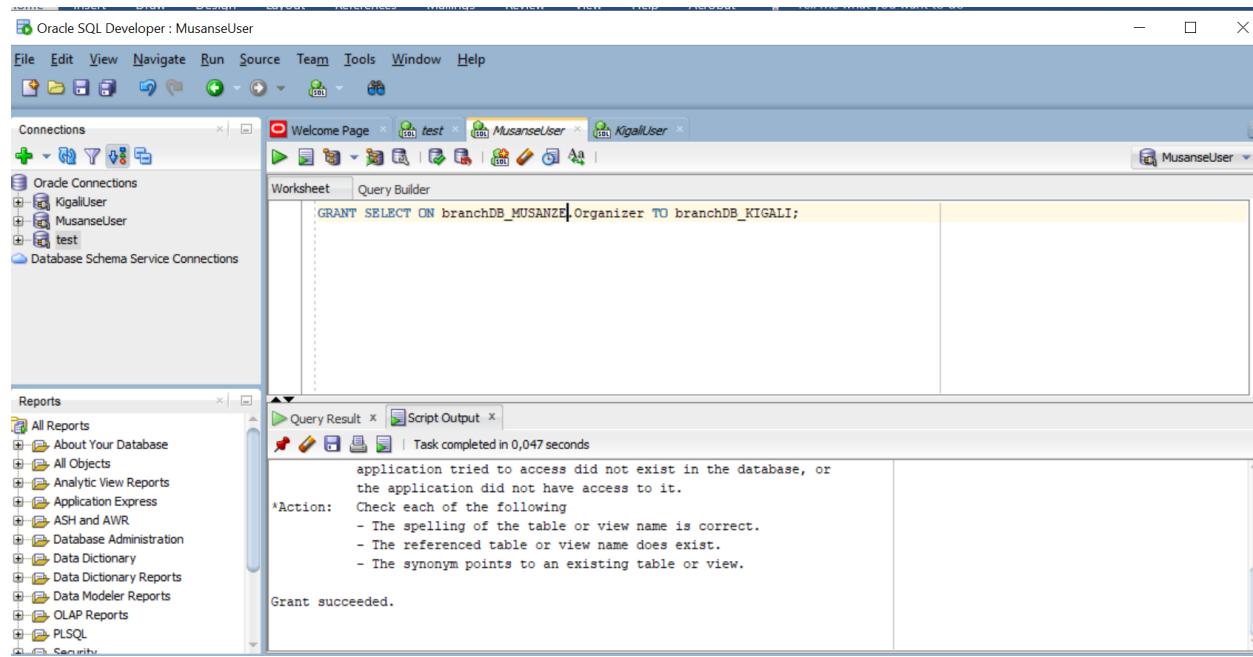
```

CREATE TABLE Payment (
PaymentID INT PRIMARY KEY,
TicketID INT,
Amount NUMBER(10,2),
Method VARCHAR2(50),
FOREIGN KEY (TicketID) REFERENCES Ticket(TicketID)
);

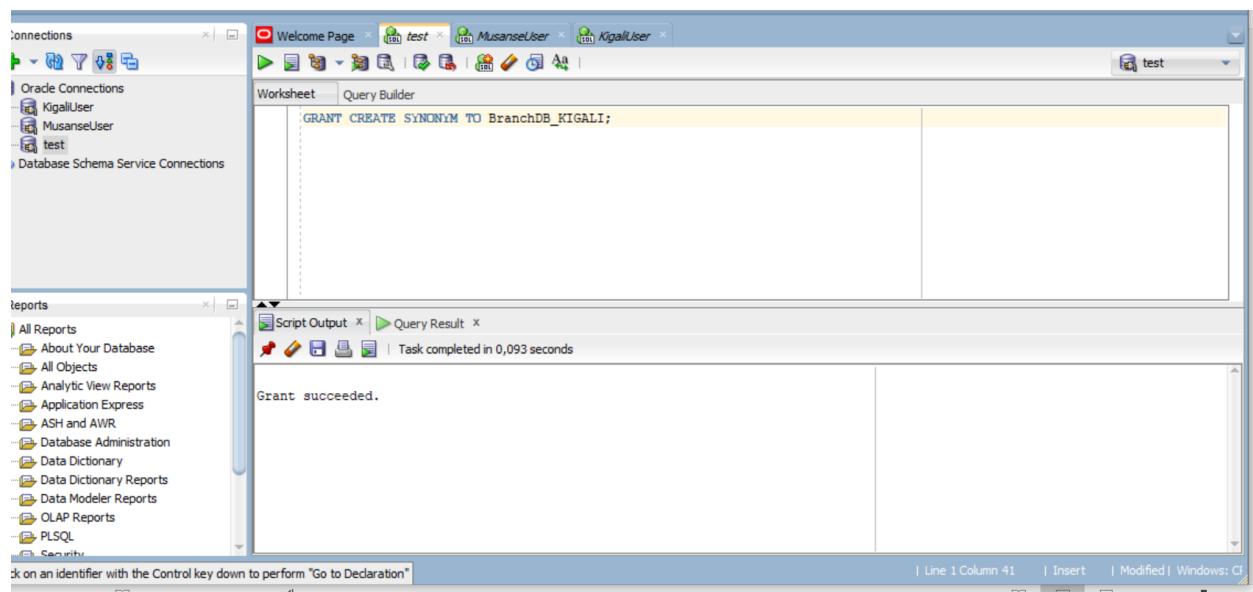
```



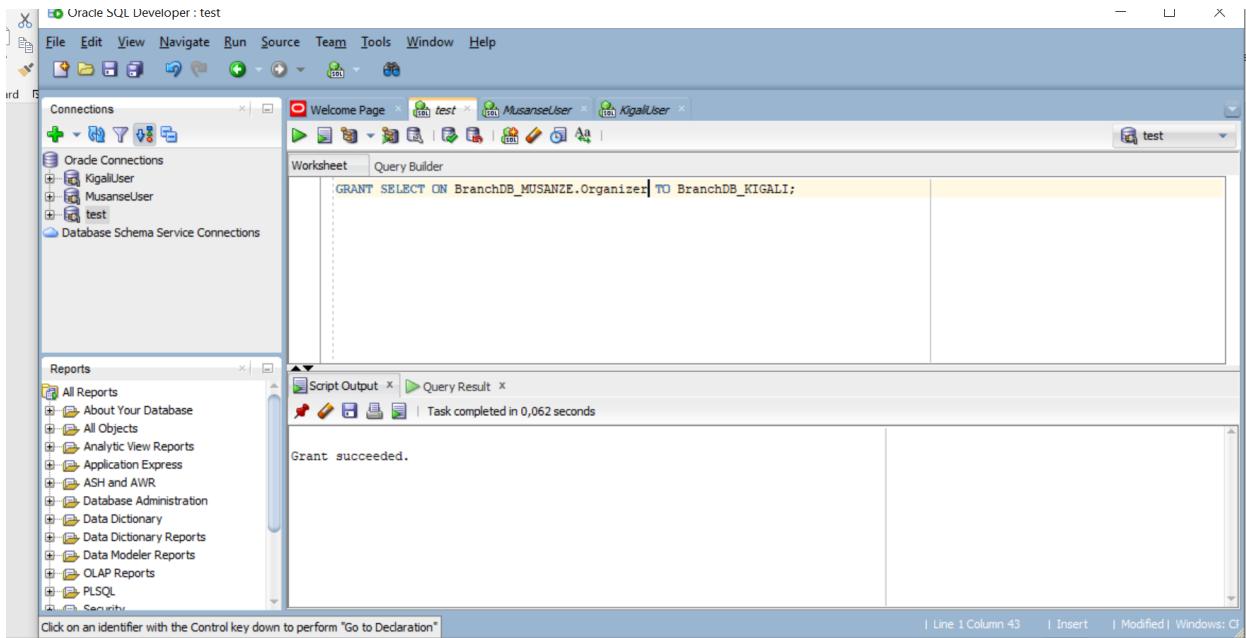
**GRANT SELECT ON branchDB\_MUSANZE.Organizer TO branchDB\_KIGALI;**



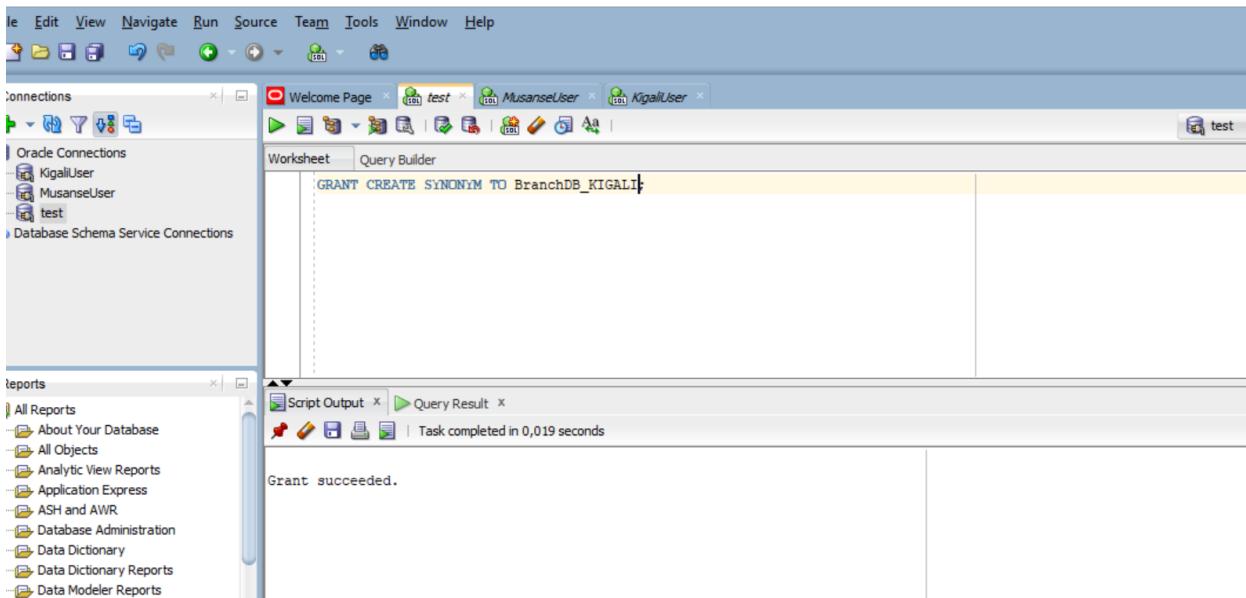
**GRANT CREATE SYNONYM TO BranchDB\_KIGALI;**



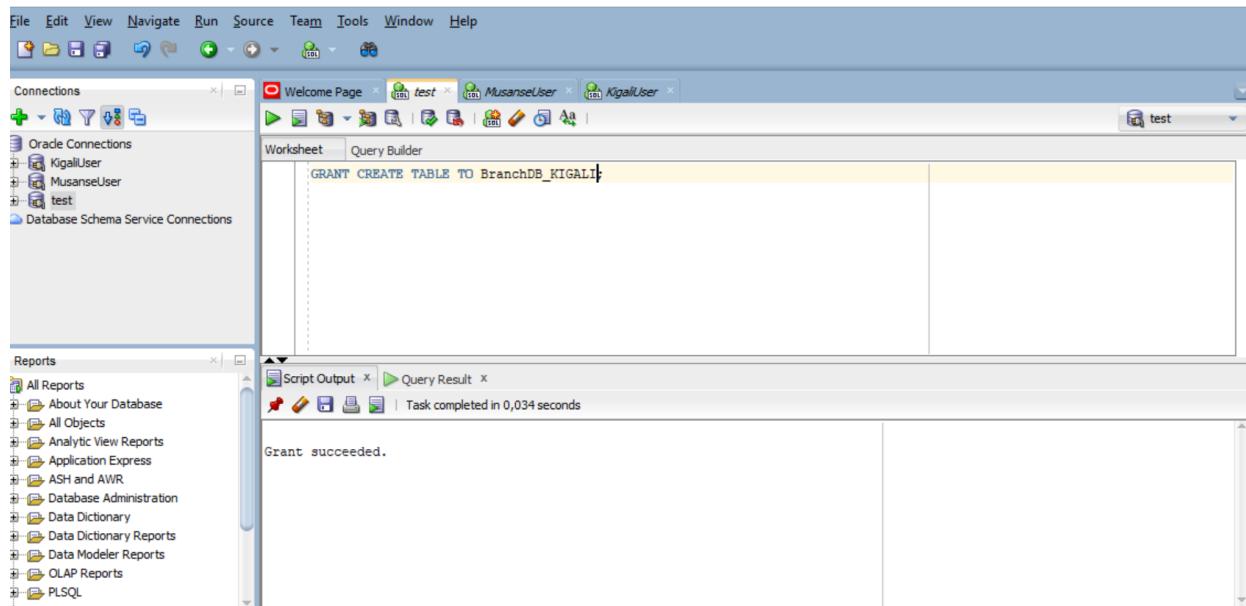
```
GRANT SELECT ON BranchDB_MUSANZE.Organizer TO BranchDB_KIGALI;
```



```
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)
);

```

The screenshot shows the Oracle SQL Developer interface. The 'Worksheet' tab contains the SQL code for creating the 'Staff' table. The 'Script Output' tab shows the confirmation message 'Table STAFF created.' The left sidebar shows connections to 'KigaliUser', 'MusaneUser', and 'test'. The bottom status bar indicates the task completed in 0,095 seconds.

```

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)
);

Table STAFF created.

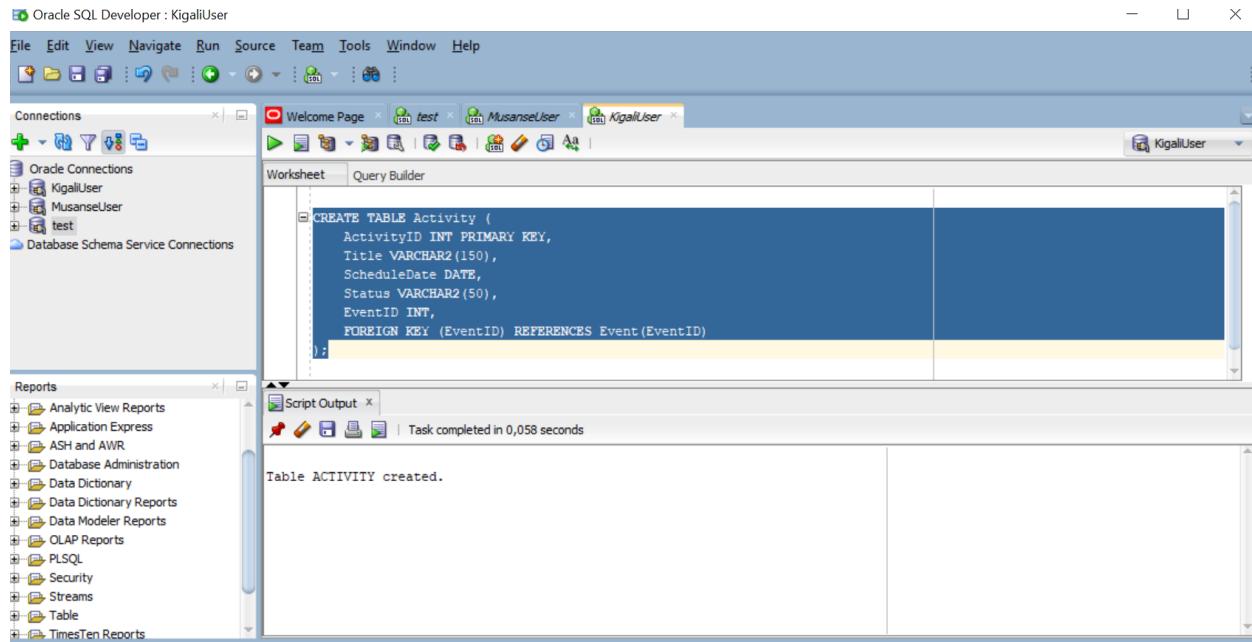
```

```

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

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)
);
```

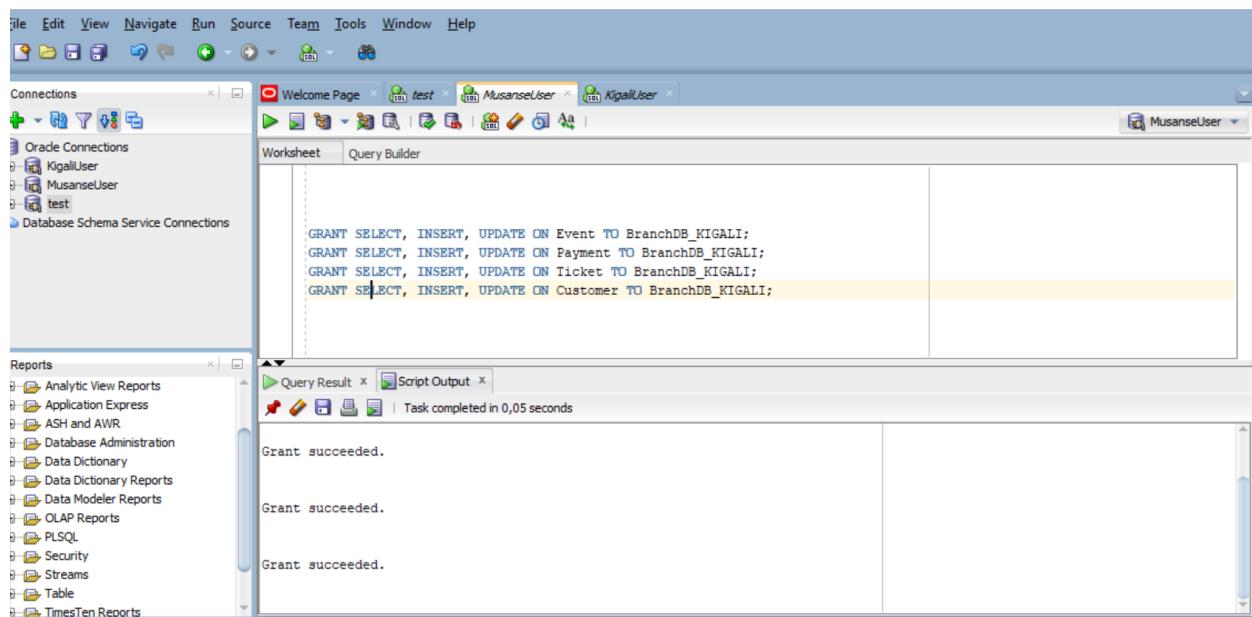
Script Output

| Task completed in 0,058 seconds

Table ACTIVITY created.

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



File Edit View Navigate Run Source Team Tools Window Help

Connections

- Oracle Connections
  - KigaliUser
  - MusanseUser
  - test
- Database Schema Service Connections

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;
```

Query Result

| Task completed in 0,05 seconds

Grant succeeded.

Grant succeeded.

Grant succeeded.

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

Oracle SQL Developer : test

```
File Edit View Navigate Run Source Team Tools Window Help
Connections
  + Oracle Connections
    KigaliUser
      Tables (Filtered)
        ACTIVITY
        CUSTOMER
        EVENT
        ORGANIZER
        PAYMENT
        REPORT
        STAFF
        TRUCK
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
  Click on an identifier with the Control key down to perform "Go to Declaration"
  Line 1 Column 2 | Insert | Modified | Windows: C
  Welcome Page test MusanzeUser KigaliUser
  Worksheet Query Builder
  ALTER USER BranchDB_KIGALI QUOTA UNLIMITED ON USERS;
  Script Output x Query Result x
  Task completed in 0,067 seconds
  User BRANCHDB_KIGALI altered.
  Line 1 Column 2 | Insert | Modified | Windows: C
```

Q2...creating link :

```
File Edit View Navigate Run Source Team Tools Window Help
Connections
  + Oracle Connections
    KigaliUser
    MusanzeUser
    test
  Database Schema Service Connections
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
  Welcome Page test MusanzeUser KigaliUser
  Worksheet Query Builder
  CREATE DATABASE LINK musanze_linknew
  CONNECT TO BranchDB_MUSANZE IDENTIFIED BY "12345"
  USING '(DESCRIPTION=
  (ADDRESS=(PROTOCOL=TCP)(HOST=192.168.43.149)(PORT=1521))
  (CONNECT_DATA=(SERVICE_NAME=XEPDB1)))
  ';
  Script Output x Query Result x
  Task completed in 0,056 seconds
  Database link MUSANZE_LINKNEW created.
  Line 1 Column 2 | Insert | Modified | Windows: C
```

SELECT TABLE FROM REMOTE BRANCHE

The screenshot shows the Oracle SQL Developer interface. On the left, the 'Connections' sidebar lists several databases: branchdb\_kigali, branchdb\_musanze, and test. Below it is the 'Reports' sidebar with various report categories. The main workspace has two tabs open: 'branchdb\_kigali12.sql' and 'branchdb\_musanze1.sql'. The 'branchdb\_kigali12.sql' tab contains the following SQL code:

```

INSERT INTO venues VALUES (101, 'Kigali Arena', 'Kigali', 5000, TO_DATE('2020-06-01', 'YYYY-MM-DD'));
INSERT INTO venues VALUES (102, 'Musanze Stadium', 'Musanze', 3000, TO_DATE('2019-04-15', 'YYYY-MM-DD'));
INSERT INTO venues VALUES (103, 'Rubavu Expo Grounds', 'Rubavu', 4000, TO_DATE('2021-08-20', 'YYYY-MM-DD'));
INSERT INTO venues VALUES (104, 'Huye Convention Center', 'Huye', 3500, TO_DATE('2022-01-10', 'YYYY-MM-DD'));
INSERT INTO venues VALUES (105, 'Nyamata Hall', 'Bugesera', 2500, TO_DATE('2023-03-05', 'YYYY-MM-DD'));

SELECT * FROM events;
SELECT * FROM customers;

```

The 'Query Result' window below shows the output of the 'customers' query:

CUSTOMERID	NAME	EMAIL	REGISTRATION_DATE
1	Alice Niyonsaba	alice@example.com	01-OCT-25
2	Eric Nkurunziza	eric@example.com	02-OCT-25
3	Diane Uwimana	diane@example.com	03-OCT-25
4	Patrick Mugisha	patrick@example.com	04-OCT-25
5	Sandrine Ishimwe	sandrine@example.com	05-OCT-25

-Distributed Join Query(events and customers fro customer\_name)

The screenshot shows the Oracle SQL Developer interface with the same connections and reports as the previous screenshot. The 'branchdb\_kigali12.sql' tab now contains a distributed join query:

```

e.eventid,
e.name AS event_name,
e.type AS event_type,
e.event_date,
c.name AS customer_name,
c.email
FROM
events e
JOIN
customers@XEFDDB1 c ON e.customerid = c.customerid;

```

The 'Query Result' window below shows the output of the joined query:

EVENTID	EVENT_NAME	EVENT_TYPE	EVENT_DATE	CUSTOMER_NAME	EMAIL
1	Tech Summit 2025	Conference	10-NOV-25	Alice Niyonsaba	alice@example.com
2	Green Innovation Fair Expo	Expo	05-DEC-25	Eric Nkurunziza	eric@example.com
3	Youth Connect Forum	Forum	20-OCT-25	Diane Uwimana	diane@example.com
4	Smart Business Meetup Networking	Networking	15-SEP-25	Patrick Mugisha	patrick@example.com
5	Creative Arts Festival Festival	Festival	30-AUG-25	Sandrine Ishimwe	sandrine@example.com

### 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

The screenshot shows the Oracle SQL Developer interface. In the top-left pane, there are connections to 'branchdb\_kigali' and 'branchdb\_musanze'. The central workspace contains two tabs: 'branchdb\_kigali12.sql' and 'branchdb\_musanze1.sql'. The 'branchdb\_musanze1.sql' tab is active and displays the following SQL code:

```
-- Serial execution (no hint)
SET TIMING ON;
EXPLAIN PLAN FOR
SELECT customerid, SUM(amount)
FROM orders
GROUP BY customerid;
|
SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);
```

Below the code, the 'Query Result' tab is selected, showing the execution plan:

PLAN_TABLE_OUTPUT						
1	Plan hash value: 2183589723	2	3	4	5	6
Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		100K	2549K	111 (8)	00:00:01
1	HASH GROUP BY		100K	2549K	111 (8)	00:00:01
2	TABLE ACCESS FULL	ORDERS	100K	2549K	105 (2)	00:00:01

### Run Parallel Query

The screenshot shows the Oracle SQL Developer interface, similar to the previous one. The 'branchdb\_musanze1.sql' tab is active and displays the following SQL code with a parallel execution hint:

```
-- Parallel execution using hint
SET TIMING ON;
EXPLAIN PLAN FOR
SELECT /*+ PARALLEL(orders, 8) */ customerid, SUM(amount)
FROM orders
GROUP BY customerid;
|
SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);
```

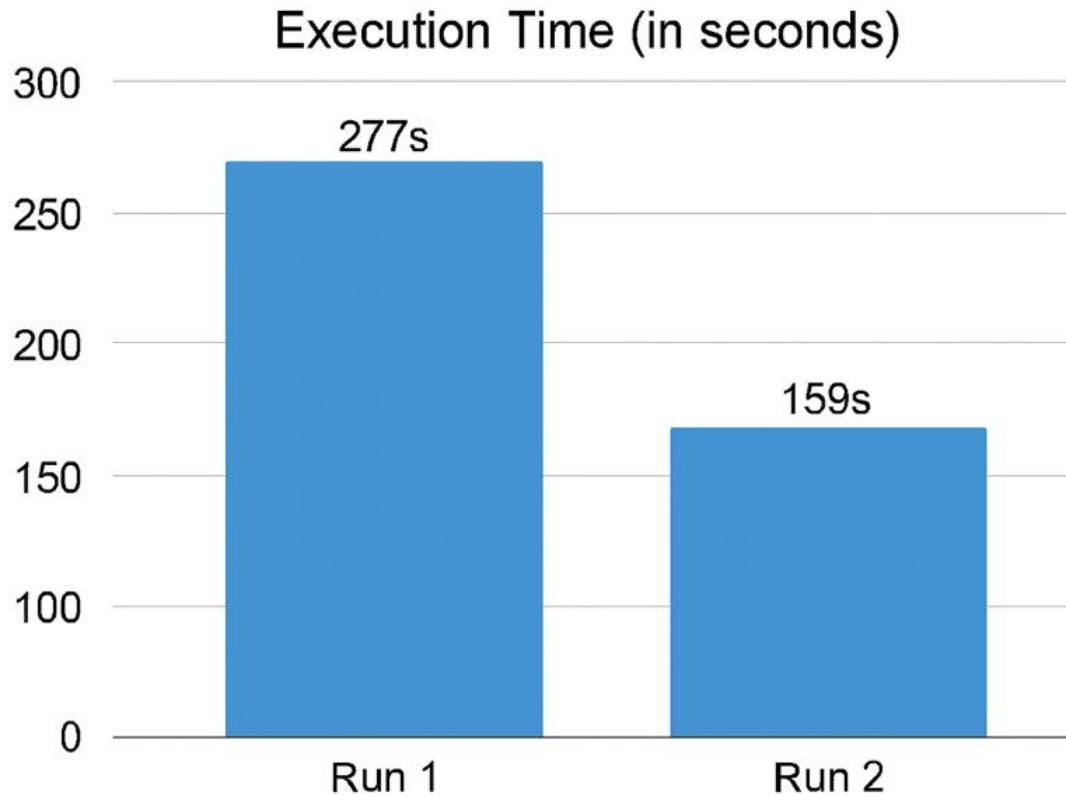
Below the code, the 'Query Result' tab is selected, showing the execution plan:

PLAN_TABLE_OUTPUT						
1	Plan hash value: 2183589723	2	3	4	5	6
Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		100K	2549K	111 (8)	00:00:01
1	HASH GROUP BY		100K	2549K	111 (8)	00:00:01
2	TABLE ACCESS FULL	ORDERS	100K	2549K	105 (2)	00:00:01

At the bottom of the plan, it indicates:

11 Hint Report (identified by operation id / Query Block Name / Object Alias):  
12 Total hints for statement: 1 (U - Unused (1))

## 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

The screenshot shows the Oracle SQL Developer interface. In the top right, there are two tabs: "branchdb\_kigali12.sql" and "branchdb\_musanze1.sql". The "branchdb\_kigali12.sql" tab is active. The "Worksheet" tab contains a PL/SQL block:

```
--distributed transaction with atomic commit using PL/SQL and verify it via DBA_2PC_PENDING.
--Step 1: PL/SQL Block for Distributed Insert

BEGIN
  -- Insert into local table
  INSERT INTO events (eventid, name, type, event_date, venueid, organizerid, customerid)
  VALUES (306, 'Digital Health Forum', 'Forum', TO_DATE('01-NOV-25','DD-MON-YY'), 106, 206, 501);

  -- Insert into remote table via DB link
  INSERT INTO customers@XEPDB1(customerid, name, email, registration_date)
```

The "Script Output" tab shows the execution results:

```
All Rows Fetched: 6 in 0,003 seconds
```

CUSTOMERID	NAME	EMAIL	REGISTRATION_DATE
1	Alice Niyonsaba	alice@example.com	01-OCT-25
2	Eric Nkurunziza	eric@example.com	02-OCT-25
3	Diane Uwimana	diane@example.com	03-OCT-25
4	Patrick Mugisha	patrick@example.com	04-OCT-25
5	Sandrine Iehimwe	sandrine@example.com	05-OCT-25
6	Jean Bosco	jean@example.com	06-OCT-25

## Step 2: Verify Atomicity Using DBA\_2PC\_PENDING

The screenshot shows the Oracle SQL Developer interface. In the top right, there are two tabs: "branchdb\_kigali12.sql" and "branchdb\_musanze1.sql". The "branchdb\_kigali12.sql" tab is active. The "Worksheet" tab contains a PL/SQL block:

```
-- Single commit
COMMIT;
END;
SELECT * FROM customers;

--Step 2: Verify Atomicity Using DBA_2PC_PENDING
SELECT LOCAL_TRAN_ID, GLOBAL_TRAN_ID, STATE, MIXED, HOST, COMMIT#
FROM DBA_2PC_PENDING;
```

The "Script Output" tab shows the execution results:

```
All Rows Fetched: 0 in 0,013 seconds
```

LOCAL_T...	GLOBAL_T...	STATE	MIXED	HOST	COMMIT#
------------	-------------	-------	-------	------	---------

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:

The screenshot shows the Oracle SQL Developer interface. In the Connections pane, there are two connections: 'branchdb\_kigali' and 'branchdb\_musanze'. The 'branchdb\_kigali' connection is selected. In the Worksheet pane, a script named 'branchdb\_musanze1.sql' is running. The code in the worksheet is:

```

BEGIN
    INSERT INTO events (eventid, name, type, event_date, venueid, organizerid, customerid)
    VALUES (307, 'AI Ethics Forum', 'Forum', TO_DATE('02-NOV-25','DD-MON-YY'), 107, 207, 502);

    INSERT INTO customers@XEADB1 (customerid, name, email, registration_date)
    VALUES (507, 'Claudine Uwase', 'claudine@example.com', TO_DATE('07-OCT-25','DD-MON-YY'));

    COMMIT;
END;
SELECT * FROM events;

```

The Query Result pane shows the results of the SELECT statement, displaying 7 rows of event data.

## 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.

The screenshot shows the Oracle SQL Developer interface. In the Connections pane, both 'branchdb\_kigali' and 'branchdb\_musanze' are listed. In the Worksheet pane, a script named 'branchdb\_kigali12.sql' is running. The code in the worksheet is:

```

-- Insert into remote Customers table via DB link
INSERT INTO customers@XEADB1 (customerid, name, email, registration_date)
VALUES (509, 'Aline Ingabire', 'aline@example.com', TO_DATE('09-OCT-25','DD-MON-YY'));

-- Commit once
COMMIT;
END;

```

An error dialog box is displayed, showing the ORA-12514 error message: "ORA-12514: Cannot connect to database. Service XEADB1 is not registered with the listener at host localhost port 1521. (CONNECTION\_ID=7ydh5z7TD1k0QWxhSTMA==) <https://docs.oracle.com/error-help/db/ora-12514/>".

The Query Result pane shows the results of the previous SELECT statement, displaying 8 rows of customer data.

## Step 4: Resolve with ROLLBACK FORCE

????????????????????????????????  
?????????????????????????????//|||||

The screenshot shows the Oracle SQL Developer interface. On the left, the Connections tree shows two connections: 'branchdb\_kigali' and 'branchdb\_musanze'. The 'branchdb\_musanze' connection is selected. The central workspace contains two tabs: 'branchdb\_kigali12.sql' and 'branchdb\_musanze1.sql'. The 'branchdb\_musanze1.sql' tab displays the following PL/SQL script:

```

-- Insert into remote Customers table via DB link
INSERT INTO customers@KEFDB1 (customerid, name, email, registration_date)
VALUES (510, 'Eric Nshimiyimana', 'eric@example.com', TO_DATE('10-OCT-25','DD-MON-YY'));

-- Commit once
COMMIT;
END;
SELECT '22.45.1234' FROM DBA_2PC_PENDING;

```

Below the script, the 'Query Result' tab shows the output of the last SELECT statement:

SID	SERIAL#	USERNAME	MACHINE	PROGRAM
1	9	61961 SYSTEM	RBC	SQL Developer

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

The screenshot shows the Oracle SQL Developer interface. The left sidebar lists various database objects and services. The central workspace contains two tabs: 'branchdb\_kigali12.sql' and 'branchdb\_musanze1.sql'. The 'branchdb\_musanze1.sql' tab displays the following PL/SQL script:

```

SELECT * FROM USER_ROLE_PRIVS;

ROLLBACK FORCE 'your_local_transaction_id';

BEGIN
    UPDATE events SET name = 'Locked by Session B' WHERE eventid = 310;
END;

```

Below the script, the 'Query Result' tab shows the output of the first SELECT statement:

Column 1	Column 2	Column 3	Column 4
Column 1 Value	Column 2 Value	Column 3 Value	Column 4 Value

The screenshot shows the Oracle SQL Developer interface. In the top-left pane, there's a tree view of database objects under 'branchdb\_jigali' and 'branchdb\_musanze'. The 'Tables (Filtered)' node is expanded. In the center, a 'Worksheet' tab is active, displaying a SQL query:

```
-- This will hang or wait due to lock conflict
END;
--Step 4: Query DBA_LOCKS to Observe the Conflict
SELECT
    l.session_id,
    s.username,
    l.lock_type,
    l.mode_held,
    l.mode_requested,
    l.blocking_others
FROM
```

Below the worksheet, the 'Query Result' tab shows the output of the query:

SESSION_ID	USERNAME	LOCK_TYPE	MODE_HELD	MODE_REQUESTED	BLOCKING_OTHERS

The status bar at the bottom indicates 'All Rows Fetched: 0 in 0,125 seconds'.

## 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

The screenshot shows the Oracle SQL Developer interface. In the top right, there's a 'branchdb\_musanze' connection icon. The main area has a 'Worksheet' tab open with the following PL/SQL code:

```

SELECT * FROM USER_ROLE_PRIVS;

ROLLBACK FORCE 'your_local_transaction_id';

BEGIN
    UPDATE events SET name = 'Locked by Session B' WHERE eventid = 310;
END;

ROLLBACK;

```

Below the worksheet, the 'Script Output' tab shows the results of the execution:

```

PL/SQL procedure successfully completed.

Elapsed: 00:06:40.526

PL/SQL procedure successfully completed.

Elapsed: 00:00:00.007

```

***PL/SQL procedure successfully completed.***

***Elapsed: 00:06:40.526***

This message 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

The screenshot shows the Oracle SQL Developer interface. In the top navigation bar, there are three tabs: 'branchdb\_kigali12.sql' (selected), 'branchdb\_musanze1.sql', and 'test'. The left sidebar contains a tree view of database objects under 'Connections' and 'Database Schema Service Connections'. The main workspace is titled 'Worksheet' and contains the following SQL code:

```
--Enable Parallel DML
ALTER SESSION ENABLE PARALLEL DML;
ALTER TABLE tickets PARALLEL 4;
--Serial Aggregation Test
SET TIMING ON
CREATE TABLE event_revenue_serial AS
SELECT eventid, SUM(price) AS total_revenue
FROM tickets
GROUP BY eventid;
```

Below the worksheet is a 'Script Output' window showing the results of the execution:

```
Table EVENT_REVENUE_SERIAL created.

Elapsed: 00:00:00.019
```

## Parallel Aggregation Test

The screenshot shows the Oracle SQL Developer interface. The top navigation bar has tabs for 'branchdb\_kigali12.sql', 'branchdb\_musanze1.sql', and 'test'. The left sidebar shows database objects. The main workspace is titled 'Worksheet' and contains the following SQL code:

```
GROUP BY eventid;

--Parallel Aggregation Test
SET TIMING ON
CREATE TABLE event_revenue_parallel AS
SELECT /*+ PARALLEL(tickets, 4) */ eventid, SUM(price) AS total_revenue
FROM tickets
GROUP BY eventid;
```

The 'Script Output' window shows the results:

```
Table EVENT_REVENUE_SERIAL created.

Elapsed: 00:00:00.019

Table EVENT_REVENUE_PARALLEL created.

Elapsed: 00:00:00.013
```

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, RESTAPIs, Java)

- Validate input
- Call local procedures `insert_customer(...)`
  - `insert_customer(...)`
  - `insert_organizer@node1(...)`
  - `insert_ticket@onode11(...)`
- Manage distributed transaction



## **Database Tier**

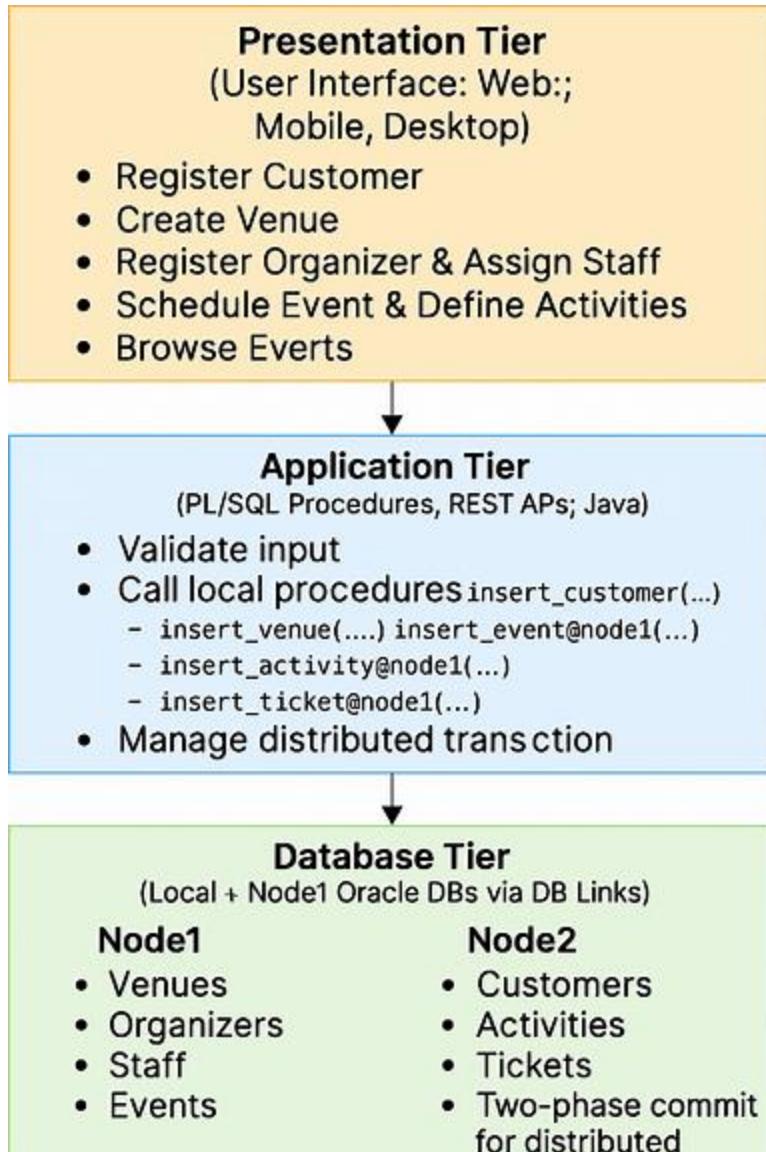
(Local + Node1 Oracle DBs via DB Links)

### **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.**

The screenshot shows two instances of Oracle SQL Developer running the same SQL query against a database named 'branchdb\_kigali12'. Both sessions are connected to the same schema and show the same results.

**Session 1 (Top):**

```

SELECT
    e.eventid,
    e.name AS event_name,
    e.type AS event_type,
    e.event_date,
    c.name AS customer_name,
    c.email
FROM
    events e
JOIN
    customers@XEPDB1 c ON e.customerid = c.customerid;
  
```

**Session 2 (Bottom):**

```

SELECT
    e.eventid,
    e.name AS event_name,
    e.type AS event_type,
    e.event_date,
    c.name AS customer_name,
    c.email
FROM
    events e
JOIN
    customers@XEPDB1 c ON e.customerid = c.customerid;
  
```

**Script Output:**

```

SQL | All Rows Fetched: 9 in 0,035 seconds

```

EVENTID	EVENT_NAME	EVENT_TYPE	EVENT_DATE	CUSTOMER_NAME	EMAIL
3	302 Green Innovation Fair	Expo	05-DEC-25	Eric Nkurunziza	eric@example.com
4	307 AI Ethics Forum	Forum	02-NOV-25	Eric Nkurunziza	eric@example.com
5	303 Youth Connect Forum	Forum	20-OCT-25	Diane Uwimana	diane@example.com
6	308 Smart Cities Expo	Expo	12-NOV-25	Diane Uwimana	diane@example.com
7	304 Smart Business Meetup	Networking	15-SEP-25	Patrick Mugisha	patrick@example.com
8	305 Creative Arts Festival	Festival	30-AUG-25	Sandrine Ishimwe	sandrine@example.com
9	310 Green Energy Summit - Conflict Summit	Conflict	20-NOV-25	Sandrine Ishimwe	sandrine@example.com

**Plan Table Output:**

```

PLAN_TABLE_OUTPUT
7 | * 1 | HASH JOIN      |   | 8 | 656 | 4 (0)| 00:00:01 |
8 | 2 | TABLE ACCESS FULL| EVENTS | 8 | 352 | 2 (0)| 00:00:01 |
9 | 3 | TABLE ACCESS FULL| CUSTOMERS | 8 | 304 | 2 (0)| 00:00:01 |
10 -----
  
```

**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.**

```

SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY());

CREATE INDEX idx_customers_customerid ON customers(customerid);

SELECT * FROM customers; -- Is it spelled correctly?
SELECT customerid, name FROM customers@XEPDB1 WHERE ROWNUM <= 5;

```

CUSTOMERID	NAME
2	502 Eric Nkurunziza
3	503 Diane Uwimana
4	504 Patrick Mugisha
5	505 Sandrine Iahimwe

## 2. Parallel Execution

```

customers c ON t.customerid = c.customerid
JOIN
payments p ON t.ticketid = p.ticketid
WHERE
e.event_date BETWEEN SYSDATE AND SYSDATE + 30;
SET AUTOTRACE ON
-- Run query
ALTER TABLE events PARALLEL 4;
ALTER TABLE tickets PARALLEL 4;
-- Run query

```

Elapsed: 00:00:00.047  
Table TICKETS altered.  
Elapsed: 00:00:00.010

## 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.