SECTION A

1Q,A.FRAGMENT AND RECOMBINE MAIN FACT

Step 1: Create Horizontal Fragments

```sql
-- On Node_A (Main Database)
CREATE TABLE Transaction_A (
    TransID SERIAL PRIMARY KEY,
    AccountID INT NOT NULL,
    TellerID VARCHAR(50) NOT NULL,
    Amount DECIMAL(15,2) NOT NULL,
    Type VARCHAR(20) CHECK (Type IN ('Credit', 'Debit')),
    DatePerformed DATE DEFAULT CURRENT_DATE
);
select * from Transaction_A;
```

```sql
-- On Node_B (Secondary Database)
CREATE TABLE Transaction_B (
    TransID SERIAL PRIMARY KEY,
    AccountID INT NOT NULL,
    TellerID VARCHAR(50) NOT NULL,
    Amount DECIMAL(15,2) NOT NULL,
    Type VARCHAR(20) CHECK (Type IN ('Credit', 'Debit')),
    DatePerformed DATE DEFAULT CURRENT_DATE
);
 select * from Transaction_B ;
```

Step 2: Insert 10 Total Rows (Split 5+5)

```sql
-- On Node_B: Insert 5 transactions (Odd AccountIDs)
INSERT INTO Transaction_B (AccountID, TellerID, Amount, Type, DatePerfor
(1001, 'T002', 2000.00, 'Credit', '2024-01-10'),
(1003, 'T003', 150.00, 'Debit', '2024-01-11'),
(1005, 'T001', 4500.75, 'Credit', '2024-01-12'),
(1007, 'T004', 750.25, 'Debit', '2024-01-13'),
(1009, 'T002', 1800.50, 'Credit', '2024-01-14');
```

Data Output Messages Notifications

Showing rows: 1 to 5 Page No: 1 of 1

	transid [PK] integer	accountid integer	tellerid character varying (50)	amount numeric (15,2)	type character varying (20)	dateperformed date
1	1	1001	T002	2000.00	Credit	2024-01-10
2	2	1003	T003	150.00	Debit	2024-01-11
3	3	1005	T001	4500.75	Credit	2024-01-12
4	4	1007	T004	750.25	Debit	2024-01-13
5	5	1009	T002	1800.50	Credit	2024-01-14

Activate Windows
Go to Settings to activate W

Step 3: Create Database Link & Unified View

```sql
-- On Node_A: Create foreign data wrapper connection to Node_B
CREATE EXTENSION IF NOT EXISTS postgres_fdw;

CREATE SERVER proj_link
FOREIGN DATA WRAPPER postgres_fdw
OPTIONS (
    host 'localhost',
    dbname 'node_b_database',
    port '5433'
);
```

```sql
CREATE USER MAPPING FOR CURRENT_USER
SERVER proj_link
OPTIONS (
    user 'postgres',
    password '12345'
);
```

```sql
-- Create the unified view
CREATE VIEW Transaction_ALL AS
SELECT TransID, AccountID, TellerID, Amount, Type, DatePerformed, 'Node_
FROM Transaction_A
UNION ALL
SELECT TransID, AccountID, TellerID, Amount, Type, DatePerformed, 'Node_
FROM Transaction_B;

select * from Transaction_ALL;
```

Data Output Messages Notifications

Showing rows: 1 to 5 Page No: 1 of 1

	transid integer	accountid integer	tellerid character varying (50)	amount numeric (15,2)	type character varying (20)	dateperformed date	sourcenode text
1	1	1002	T001	1500.00	Credit	2024-01-15	Node_A
2	2	1004	T002	250.75	Debit	2024-01-16	Node_A
3	3	1006	T003	3000.00	Credit	2024-01-17	Node_A
4	4	1008	T001	500.50	Debit	2024-01-18	Node_A
5	5	1010	T004	1200.25	Credit	2024-01-19	Node_A

Activate Windows

Step 4: Validation with COUNT and Checksum

```sql
-- Validate fragmentation and recombination
SELECT
    'Transaction_A' as Table_Name,
    COUNT(*) as Row_Count,
    SUM(MOD(TransID, 97)) as Checksum
FROM Transaction_A
UNION ALL
SELECT
    'Transaction_B' as Table_Name,
    COUNT(*) as Row_Count,
    SUM(MOD(TransID, 97)) as Checksum
FROM Transaction_B
UNION ALL
SELECT
    'Transaction_ALL' as Table_Name,
    COUNT(*) as Row_Count,
    SUM(MOD(TransID, 97)) as Checksum
FROM Transaction_ALL;
```

Data Output Messages Notifications

Showing rows: 1 to 3 Page No: 1

	table_name text	row_count bigint	checksum bigint
1	Transaction_A	5	15
2	Transaction_B	0	[null]
3	Transaction_ALL	5	15

Step 5: Verify Data Distribution

```sql
-- Check data distribution across nodes
SELECT
    SourceNode,
    COUNT(*) as TransactionCount,
    SUM(Amount) as TotalAmount,
    AVG(Amount) as AverageAmount
FROM Transaction_ALL
GROUP BY SourceNode
ORDER BY SourceNode;
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No:

	sourcenode text	transactioncount bigint	totalamount numeric	averageamount numeric
1	Node_A	5	6451.50	1290.3000000000000000

```sql
-- Verify individual fragments
SELECT 'Node_A Data' as Info, * FROM Transaction_A ORDER BY TransID;
SELECT 'Node_B Data' as Info, * FROM Transaction_B ORDER BY TransID;


-- Expected Output:
-- Table_Name     | Row_Count | Checksum
-- Transaction_A  | 5         | [sum of mod values]
-- Transaction_B  | 5         | [sum of mod values]
-- Transaction_ALL| 10        | [sum of both above]
```

Data Output Messages Notifications

Showing rows: 1 to 5 Page No: 1 of 1

	info text	transid [PK] integer	accountid integer	tellerid character varying (50)	amount numeric (15,2)	type character varying (20)	dateperformed date
1	Node_A Data	1	1002	T001	1500.00	Credit	2024-01-15
2	Node_A Data	2	1004	T002	250.75	Debit	2024-01-16
3	Node_A Data	3	1006	T003	3000.00	Credit	2024-01-17
4	Node_A Data	4	1008	T001	500.50	Debit	2024-01-18
5	Node_A Data	5	1010	T004	1200.25	Credit	2024-01-19

Activate Windows

Q2.

Step 2: Insert 10 Committed Rows (5+5 Split)

On Node_A: Insert 5 Transactions

```sql
-- Node_A Transactions (Even AccountIDs: 1002, 1004, 1006, 1008, 1010)
INSERT INTO Transaction_A (AccountID, TellerID, Amount, Type, DatePerfor
(1002, 'T001', 1500.00, 'Credit', '2024-01-15'),
(1004, 'T002', 250.75, 'Debit', '2024-01-16'),
(1006, 'T003', 3000.00, 'Credit', '2024-01-17'),
(1008, 'T001', 500.50, 'Debit', '2024-01-18'),
(1010, 'T004', 1200.25, 'Credit', '2024-01-19');

-- Verify Node_A insertion
SELECT 'Node_A inserted: ' || COUNT(*)::TEXT || ' rows' FROM Transactior
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	?column? text
1	Node_A inserted: 10 rows

```sql
-- Node_B Transactions (Odd AccountIDs: 1001, 1003, 1005, 1007, 1009)
INSERT INTO Transaction_B (AccountID, TellerID, Amount, Type, DatePerfor
(1001, 'T002', 2000.00, 'Credit', '2024-01-10'),
(1003, 'T003', 150.00, 'Debit', '2024-01-11'),
(1005, 'T001', 4500.75, 'Credit', '2024-01-12'),
(1007, 'T004', 750.25, 'Debit', '2024-01-13'),
(1009, 'T002', 1800.50, 'Credit', '2024-01-14');

-- Verify Node_B insertion
SELECT 'Node_B inserted: ' || COUNT(*)::TEXT || ' rows' FROM Transactior
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	?column? text
1	Node_B inserted: 0 rows

Complete Validation Script

```sql
--------Complete Validation Script
-- Comprehensive validation to ensure exactly 10 committed rows
WITH counts AS (
    SELECT
        (SELECT COUNT(*) FROM Transaction_A) as count_a,
        (SELECT COUNT(*) FROM Transaction_B) as count_b
)
SELECT
    'Validation Results:' as check_type,
    'Node_A rows: ' || count_a::TEXT as node_a_count,
    'Node_B rows: ' || count_b::TEXT as node_b_count,
    'Total rows: ' || (count_a + count_b)::TEXT as total_count,
    CASE
        WHEN (count_a + count_b) = 10 THEN '✅ PASS: Exactly 10 rows to
        ELSE '❌ FAIL: Row count mismatch'
    END as status
FROM counts;
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No:

	check_type text	node_a_count text	node_b_count text	total_count text	status text
1	Validation Results:	Node_A rows: 10	Node_B rows: 0	Total rows: 10	✅ PASS: Exactly 10 rows total

```sql
-- Show all data with source identification
SELECT 'Node_A Data' as source, TransID, AccountID, TellerID, Amount, Ty
FROM Transaction_A
UNION ALL
SELECT 'Node_B Data' as source, TransID, AccountID, TellerID, Amount, Ty
FROM Transaction_B
ORDER BY source, TransID;
```

Data Output Messages Notifications

Showing rows: 1 to 10 Page No: 1 of 1

	source text	transid integer	accountid integer	tellerid character varying (50)	amount numeric (15,2)	type character varying (20)	dateperformed date
1	Node_A Data	1	1002	T001	1500.00	Credit	2024-01-15
2	Node_A Data	2	1004	T002	250.75	Debit	2024-01-16
3	Node_A Data	3	1006	T003	3000.00	Credit	2024-01-17
4	Node_A Data	4	1008	T001	500.50	Debit	2024-01-18
5	Node_A Data	5	1010	T004	1200.25	Credit	2024-01-19
6	Node_A Data	6	1002	T001	1500.00	Credit	2024-01-15

Activate Windows
Go to Settings to activate Windows.

Data Distribution Analysis

```sql
-- Analyze the data distribution pattern
SELECT
    'Even AccountIDs on Node_A' as distribution_rule,
    COUNT(*) as transaction_count,
    SUM(Amount) as total_amount,
    AVG(Amount) as average_amount
FROM Transaction_A
UNION ALL
SELECT
    'Odd AccountIDs on Node_B' as distribution_rule,
```

```sql
    COUNT(*) as transaction_count,
    SUM(Amount) as total_amount,
    AVG(Amount) as average_amount
FROM Transaction_B;
```

Data Output Messages Notifications

Showing rows: 1 to 2 Page No:

	distribution_rule text	transaction_count bigint	total_amount numeric	average_amount numeric
1	Even AccountIDs on Node_A	10	12903.00	1290.3000000000000000
2	Odd AccountIDs on Node_B	0	[null]	[null]

```sql
erify fragmentation logic
CT
'AccountID Pattern Check' as check_type,
'Node_A (Even IDs): ' ||
(SELECT COUNT(*) FROM Transaction_A WHERE AccountID % 2 = 0)::TEXT || '
'Node_B (Odd IDs): ' ||
(SELECT COUNT(*) FROM Transaction_B WHERE AccountID % 2 = 1)::TEXT || '
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1

	check_type text	node_a_pattern text	node_b_pattern text
1	AccountID Pattern Check	Node_A (Even IDs): 10 rows	Node_B (Odd IDs): 0 rows

Q3.

Create Unified View with Database Link

First, Ensure Database Link is Properly Configured

```sql
-- Verify the server is created
SELECT srvname, srvoptions FROM pg_foreign_server WHERE srvname = 'proj_
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1

	srvname name	srvoptions text[]
1	proj_link	{host=localhost,dbname=BankingTransactionB,port=5433}

Import Transaction_B as Foreign Table

```sql
-- Verify foreign table was created
SELECT
    ft.ftrelid::regclass as foreign_table,
    fs.srvname as foreign_server
FROM pg_foreign_table ft
JOIN pg_foreign_server fs ON ft.ftserver = fs.oid
WHERE ft.ftrelid::regclass = 'transaction_b'::regclass;
```

Data Output Messages Notifications

foreign_table regclass	foreign_server name

```sql
SELECT
    TransID,
    AccountID,
    TellerID,
    Amount,
    Type,
    DatePerformed,
    'Node_B' as SourceNode
FROM transaction_b;  -- This is the foreign table from Node_B
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	table_name name	view_definition character varying
1	transaction_all	SELECT transaction_a.transid, transaction_a.accountid, transaction_a.tellerid, transaction_a.amount, transaction_a.type, transaction_

Test the Unified View

```sql

-- Test 1: Count all records from the unified view
SELECT
    'Transaction_ALL View Test' as test_type,
    COUNT(*) as total_rows,
    COUNT(DISTINCT SourceNode) as nodes_accessed
FROM Transaction_ALL;
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	test_type text	total_rows bigint	nodes_accessed bigint
1	Transaction_ALL View Test	10	1

```sql
-- Test 2: Verify data from both nodes appears
SELECT
    SourceNode,
    COUNT(*) as row_count,
    SUM(Amount) as total_amount,
    MIN(DatePerformed) as earliest_date,
    MAX(DatePerformed) as latest_date
FROM Transaction_ALL
GROUP BY SourceNode
ORDER BY SourceNode;
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	sourcenode text	row_count bigint	total_amount numeric	earliest_date date	latest_date date
1	Node_A	10	12903.00	2024-01-15	2024-01-19

```sql
-- Test 3: Show sample data from the unified view
SELECT
    TransID,
    AccountID,
    TellerID,
    Amount,
    Type,
    DatePerformed,
    SourceNode
FROM Transaction_ALL
```

Data Output Messages Notifications

Showing rows: 1 to 10 Page No: 1 of 1

	transid integer	accountid integer	tellerid character varying (50)	amount numeric (15,2)	type character varying (20)	dateperformed date	sourcenode text
1	1	1002	T001	1500.00	Credit	2024-01-15	Node_A
2	2	1004	T002	250.75	Debit	2024-01-16	Node_A
3	3	1006	T003	3000.00	Credit	2024-01-17	Node_A
4	4	1008	T001	500.50	Debit	2024-01-18	Node_A
5	5	1010	T004	1200.25	Credit	2024-01-19	Node_A
6	6	1002	T001	1500.00	Credit	2024-01-15	Node_A

Activate Windows
Go to Settings to activate Windows.

```sql
-- Verify the view was created
SELECT table_name, view_definition
FROM information_schema.views
WHERE table_name = 'transaction_all';
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	table_name name	view_definition character varying
1	transaction_all	SELECT transaction_a.transid, transaction_a.accountid, transaction_a.tellerid, transaction_a.amount, transaction_a.type, transaction_

Comprehensive Validation

```sql
    fs.fragment_count as rows_in_fragments,
    vs.view_total as amount_in_view,
    fs.fragment_total as amount_in_fragments,
    CASE
        WHEN vs.view_count = fs.fragment_count
        AND vs.view_total = fs.fragment_total
        THEN '✅ PASS: View matches fragment totals'
        ELSE '❌ FAIL: Data mismatch detected'
    END as validation_result
FROM view_stats vs, fragment_stats fs;
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	check_type text	rows_in_view bigint	rows_in_fragments bigint	amount_in_view numeric	amount_in_fragments numeric	validation_result text
1	Data Consistency Check	10	10	12903.00	12903.00	✅ PASS: View matches fragment totals

Test Cross-Node Queries

```sql
-- Query 1: Find top 3 largest transactions regardless of node
SELECT
    TransID,
    AccountID,
    TellerID,
    Amount,
    Type,
    DatePerformed,
    SourceNode
FROM Transaction_ALL
```

Data Output Messages Notifications

Showing rows: 1 to 3 Page No: 1 of 1

	transid integer	accountid integer	tellerid character varying (50)	amount numeric (15,2)	type character varying (20)	dateperformed date	sourcenode text
1	8	1006	T003	3000.00	Credit	2024-01-17	Node_A
2	3	1006	T003	3000.00	Credit	2024-01-17	Node_A
3	1	1002	T001	1500.00	Credit	2024-01-15	Node_A

```sql
-- Query 2: Daily transaction summary across both nodes
SELECT
    DatePerformed,
    COUNT(*) as transaction_count,
    SUM(Amount) as daily_total,
    COUNT(DISTINCT SourceNode) as nodes_used
FROM Transaction_ALL
GROUP BY DatePerformed
ORDER BY DatePerformed;
```

Data Output Messages Notifications

Showing rows: 1 to 5 Page No: 1 of 1

	dateperformed date	transaction_count bigint	daily_total numeric	nodes_used bigint
1	2024-01-15	2	3000.00	1
2	2024-01-16	2	501.50	1
3	2024-01-17	2	6000.00	1
4	2024-01-18	2	1001.00	1
5	2024-01-19	2	2400.50	1

Activate Windows

```sql
-- Query 3: Teller performance across distributed system
SELECT
    TellerID,
    COUNT(*) as transactions_handled,
    SUM(CASE WHEN Type = 'Credit' THEN Amount ELSE 0 END) as total_cred
    SUM(CASE WHEN Type = 'Debit' THEN Amount ELSE 0 END) as total_debits
    COUNT(DISTINCT SourceNode) as nodes_worked_on
FROM Transaction_ALL
GROUP BY TellerID
ORDER BY transactions_handled DESC;
```

Data Output Messages Notifications

Showing rows: 1 to 4 Page No: 1

	tellerid character varying (50)	transactions_handled bigint	total_credits numeric	total_debits numeric	nodes_worked_on bigint
1	T001	4	3000.00	1001.00	1
2	T002	2	0	501.50	1
3	T003	2	6000.00	0	1
4	T004	2	2400.50	0	1

Troubleshooting Commands

If you encounter issues, run these diagnostic commands:

```sql
SELECT * FROM transaction_b LIMIT 1;

-- Check foreign server status
SELECT
    srvname,
    srvoptions,
    (SELECT count(*) FROM information_schema.tables WHERE table_schema =
    (SELECT count(*) FROM information_schema.foreign_tables) as foreign_
FROM pg_foreign_server
WHERE srvname = 'proj_link';
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1

	srvname name	srvoptions text[]	local_tables bigint	foreign_tables bigint
1	proj_link	{host=localhost,dbname=BankingTransactionB,port=5433}	3	0

```
    (SELECT count(*) FROM information_schema.foreign_tables) as foreign_
FROM pg_foreign_server
WHERE srvname = 'proj_link';

-- Test basic UNION ALL without view
SELECT 'Node_A' as source, * FROM Transaction_A
UNION ALL
SELECT 'Node_B' as source, * FROM transaction_b
ORDER BY source, TransID;
```

Data Output Messages Notifications

Showing rows: 1 to 10 Page No: 1 of 1

	source text	transid integer	accountid integer	tellerid character varying (50)	amount numeric (15,2)	type character varying (20)	dateperformed date
1	Node_A	1	1002	T001	1500.00	Credit	2024-01-15
2	Node_A	2	1004	T002	250.75	Debit	2024-01-16
3	Node_A	3	1006	T003	3000.00	Credit	2024-01-17
4	Node_A	4	1008	T001	500.50	Debit	2024-01-18
5	Node_A	5	1010	T004	1200.25	Credit	2024-01-19
6	Node_A	6	1002	T001	1500.00	Credit	2024-01-15

Activate Windows
Go to Settings to activate Windows

Q4.

```
----------

-- DETAILED VALIDATION WITH EVIDENCE
SELECT '=== COMPLETE VALIDATION REPORT ===' as report_header;
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	report_header text
1	=== COMPLETE VALIDATION REPORT ===

```sql
-- 1. Count Validation
SELECT
    '1. ROW COUNT VALIDATION' as test_type,
    (SELECT COUNT(*) FROM Transaction_A) as node_a_count,
    (SELECT COUNT(*) FROM transaction_b) as node_b_count,
    (SELECT COUNT(*) FROM Transaction_ALL) as view_count,
    CASE
        WHEN (SELECT COUNT(*) FROM Transaction_ALL) =
              (SELECT COUNT(*) FROM Transaction_A) +
              (SELECT COUNT(*) FROM transaction_b)
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1

	test_type text	node_a_count bigint	node_b_count bigint	view_count bigint	count_status text
1	1. ROW COUNT VALIDATION	10	0	10	PASS: Row counts match

```sql
-- 2. Checksum Validation
SELECT
    '2. CHECKSUM VALIDATION (MOD(TransID, 97))' as test_type,
    (SELECT SUM(MOD(TransID, 97)) FROM Transaction_A) as node_a_checksur
    (SELECT SUM(MOD(TransID, 97)) FROM transaction_b) as node_b_checksur
    (SELECT SUM(MOD(TransID, 97)) FROM Transaction_ALL) as view_checksur
    CASE
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	test_type text	node_a_checksum bigint	node_b_checksum bigint	view_checksum bigint	checksum_status text
1	2. CHECKSUM VALIDATION (MOD(TransID, 97))	55	[null]	55	FAIL: Checksum mismatch

```sql
-- 3. Data Integrity Validation
SELECT
    '3. DATA INTEGRITY VALIDATION' as test_type,
    (SELECT SUM(Amount) FROM Transaction_A) as node_a_total,
    (SELECT SUM(Amount) FROM transaction_b) as node_b_total,
    (SELECT SUM(Amount) FROM Transaction_ALL) as view_total,
    CASE
        WHEN (SELECT SUM(Amount) FROM Transaction_ALL) =
              (SELECT SUM(Amount) FROM Transaction_A) +
              (SELECT SUM(Amount) FROM transaction_b)
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of

	test_type text	node_a_total numeric	node_b_total numeric	view_total numeric	integrity_status text
1	3. DATA INTEGRITY VALIDATION	12903.00	[null]	12903.00	FAIL: Data total mismatch

```sql
-- 4. Final Comprehensive Result
SELECT
    'FINAL VALIDATION RESULT' as result_type,
    CASE
        WHEN (SELECT COUNT(*) FROM Transaction_ALL) = 10
        AND (SELECT SUM(MOD(TransID, 97)) FROM Transaction_ALL) =
            (SELECT SUM(MOD(TransID, 97)) FROM Transaction_A) +
            (SELECT SUM(MOD(TransID, 97)) FROM transaction_b)
        AND (SELECT SUM(Amount) FROM Transaction_ALL) =
            (SELECT SUM(Amount) FROM Transaction_A) +
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	result_type text	final_result text
1	FINAL VALIDATION RESULT	FAILURE: Data consistency issues detected

Evidence Collection for Screenshot

```sql
    ------------------Evidence Collection for Screenshot-- CLEAN EVIDEN
SELECT 'EVIDENCE: Fragment vs Unified View Comparison' as title;

-- Fragment Data
SELECT 'FRAGMENT_DATA' as data_type, 'Transaction_A' as source, * FROM
UNION ALL
SELECT 'FRAGMENT_DATA' as data_type, 'Transaction_B' as source, * FROM
ORDER BY source, TransID;
```

Data Output Messages Notifications

Showing rows: 1 to 10 Page No: 1 of 1

	data_type text	source text	transid integer	accountid integer	tellerid character varying (50)	amount numeric (15,2)	type character varying (20)	dateperformed date
1	FRAGMENT_DATA	Transaction_A	1	1002	T001	1500.00	Credit	2024-01-15
2	FRAGMENT_DATA	Transaction_A	2	1004	T002	250.75	Debit	2024-01-16
3	FRAGMENT_DATA	Transaction_A	3	1006	T003	3000.00	Credit	2024-01-17
4	FRAGMENT_DATA	Transaction_A	4	1008	T001	500.50	Debit	2024-01-18
5	FRAGMENT_DATA	Transaction_A	5	1010	T004	1200.25	Credit	2024-01-19

Activate Windows
Go to Settings to activate Windows.

```sql
-- Validation Summary (Perfect for screenshot)
SELECT
    'SUMMARY: Count and Checksum Validation' as validation_type,
    'Expected: 10 rows total' as expectation,
    'Actual: ' || (SELECT COUNT(*) FROM Transaction_ALL)::TEXT || ' rows
    'Checksum (A): ' || (SELECT SUM(MOD(TransID, 97)) FROM Transaction_A
    'Checksum (B): ' || (SELECT SUM(MOD(TransID, 97)) FROM transaction_b
    'Checksum (ALL): ' || (SELECT SUM(MOD(TransID, 97)) FROM Transaction
    CASE
        WHEN (SELECT COUNT(*) FROM Transaction_ALL) = 10
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	validation_type text	expectation text	actual_count text	checksum_a text	checksum_b text	checksum_all text	result text
1	SUMMARY: Count and Checksum Validation	Expected: 10 rows total	Actual: 10 rows	Checksum (A): 55	[null]	Checksum (ALL): 55	✗ VAI

```sql
-- Fragment Data
SELECT 'FRAGMENT_DATA' as data_type, 'Transaction_A' as source, * FROM T
UNION ALL
SELECT 'FRAGMENT_DATA' as data_type, 'Transaction_B' as source, * FROM t
ORDER BY source, TransID;

-- Validation Summary (Perfect for screenshot)
SELECT
    'SUMMARY: Count and Checksum Validation' as validation_type,
    'Expected: 10 rows total' as expectation,
```

Data Output Messages Notifications

Showing rows: 1 to 10 Page No: 1 of 1

	data_type text	source text	transid integer	accountid integer	tellerid character varying (50)	amount numeric (15,2)	type character varying (20)	dateperformed date
1	FRAGMENT_DATA	Transaction_A	1	1002	T001	1500.00	Credit	2024-01-15
2	FRAGMENT_DATA	Transaction_A	2	1004	T002	250.75	Debit	2024-01-16
3	FRAGMENT_DATA	Transaction_A	3	1006	T003	3000.00	Credit	2024-01-17
4	FRAGMENT_DATA	Transaction_A	4	1008	T001	500.50	Debit	2024-01-18
5	FRAGMENT_DATA	Transaction_A	5	1010	T004	1200.25	Credit	2024-01-19

Activate Windows
Go to Settings to activate Windows.

Complete DDL and Population Scripts Recap

```sql
-- Transaction_A DDL
SELECT 'CREATE TABLE Transaction_A (' as ddl_line
UNION ALL SELECT '    TransID SERIAL PRIMARY KEY,'
UNION ALL SELECT '    AccountID INT NOT NULL,'
UNION ALL SELECT '    TellerID VARCHAR(50) NOT NULL,'
UNION ALL SELECT '    Amount DECIMAL(15,2) NOT NULL,'
UNION ALL SELECT '    Type VARCHAR(20) CHECK (Type IN (''Credit'', ''Deb
UNION ALL SELECT '    DatePerformed DATE DEFAULT CURRENT_DATE'
UNION ALL SELECT ');';
```

Data Output | Messages | Notifications

Showing rows: 1 to 8 | Page No: 1 of 1

	ddl_line text
1	CREATE TABLE Transaction_A (
2	TransID SERIAL PRIMARY KEY,
3	AccountID INT NOT NULL,
4	TellerID VARCHAR(50) NOT NULL,
5	Amount DECIMAL(15,2) NOT NULL,
6	Type VARCHAR(20) CHECK (Type IN ('Credit', 'Deb...

Activate Windows
Go to Settings to activate Windows.

```sql
-- Transaction_B DDL
SELECT 'CREATE TABLE Transaction_B (' as ddl_line
UNION ALL SELECT '    TransID SERIAL PRIMARY KEY,'
UNION ALL SELECT '    AccountID INT NOT NULL,'
UNION ALL SELECT '    TellerID VARCHAR(50) NOT NULL,'
UNION ALL SELECT '    Amount DECIMAL(15,2) NOT NULL,'
UNION ALL SELECT '    Type VARCHAR(20) CHECK (Type IN (''Credit'', ''Deb
UNION ALL SELECT '    DatePerformed DATE DEFAULT CURRENT_DATE'
UNION ALL SELECT ');';
```

Data Output | Messages | Notifications

Showing rows: 1 to 8 | Page No: 1

	ddl_line text
1	CREATE TABLE Transaction_B (
2	TransID SERIAL PRIMARY KEY,
3	AccountID INT NOT NULL,
4	TellerID VARCHAR(50) NOT NULL,
5	Amount DECIMAL(15,2) NOT NULL,
6	Type VARCHAR(20) CHECK (Type IN ('Credit', 'Deb...

Activate W
Go to Settings

Database Link Creation Recap

```sql
-- DATABASE LINK CREATION (Recap)
SELECT '=== DATABASE LINK CREATION ===' as link_section;

SELECT 'CREATE SERVER proj_link' as link_ddl
UNION ALL SELECT 'FOREIGN DATA WRAPPER postgres_fdw'
UNION ALL SELECT 'OPTIONS ('
UNION ALL SELECT '    host ''localhost'','
UNION ALL SELECT '    dbname ''BankingTransactionB'','
UNION ALL SELECT '    port ''5433'''
```

Data Output Messages Notifications

Showing rows: 1 to 14 Page No: 1 of 1

	link_ddl text
1	CREATE SERVER proj_link
2	FOREIGN DATA WRAPPER postgres_fdw
3	OPTIONS (
4	host 'localhost',
5	dbname 'BankingTransactionB',
6	port '5433'

Activate Windows
Go to Settings to activate Windows.

Unified View Creation Recap

```sql
--------Unified View Creation Recap
-- UNIFIED VIEW CREATION (Recap)
SELECT '=== UNIFIED VIEW CREATION ===' as view_section;

SELECT 'CREATE VIEW Transaction_ALL AS' as view_ddl
UNION ALL SELECT 'SELECT'
UNION ALL SELECT '    TransID,'
UNION ALL SELECT '    AccountID,'
UNION ALL SELECT '    TellerID,'
UNION ALL SELECT '    Amount,'
UNION ALL SELECT '    Type,'
```

Data Output Messages Notifications

Showing rows: 1 to 20 Page No: 1 of 1

	view_ddl text
1	CREATE VIEW Transaction_ALL AS
2	SELECT
3	TransID,
4	AccountID,
5	TellerID,
6	Amount,

Activate Windows
Go to Settings to activate Windows.

A2: Database Link & Cross-Node Join (3-10 rows result)

Step 1: Create Database Link to Node_B

Q1.

```
);

-- Verify the database link
SELECT
    srvname as server_name,
    srvoptions as connection_details
FROM pg_foreign_server
WHERE srvname = 'proj_link';
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	server_name name	connection_details text[]
1	proj_link	{host=localhost,dbname=BankingTransactionB,port=5433}

Q2:Remote SELECT on Account@proj_link

Step 1: Import Account Table from Node_B

```
-- Sample 3: Account summary with statistics
SELECT '=== ACCOUNT SUMMARY STATISTICS (Sample 3) ===' as query_title;
SELECT
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	query_title text
1	=== ACCOUNT SUMMARY STATISTICS (Sample 3) ===

```sql
-- Check what tables are available on the remote server
SELECT
    'Available Remote Tables' as info,
    table_name,
    table_type
FROM information_schema.tables
WHERE table_schema = 'public'
ORDER BY table_name;
```

Data Output Messages Notifications

Showing rows: 1 to 4 Page No: 1 of 1

	info text	table_name name	table_type character varying
1	Available Remote Tables	remote_loan	FOREIGN
2	Available Remote Tables	transaction_a	BASE TABLE
3	Available Remote Tables	transaction_all	VIEW
4	Available Remote Tables	transaction_b	BASE TABLE

Activate Windows

```sql
-- Check foreign data wrapper status
SELECT
    'FDW Status' as status_check,
    srvname as server_name,
    srvoptions as connection_details
FROM pg_foreign_server;
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	status_check text	server_name name	connection_details text[]
1	FDW Status	proj_link	{host=localhost,dbname=BankingTransactionB,port=5433}

Q3.Step 1: Import Teller Table from Node_B

```sql
-- EVIDENCE 1: Database Link Configuration
SELECT '=== DATABASE LINK CONFIGURATION ===' as evidence_section;
SELECT
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	evidence_section text
1	=== DATABASE LINK CONFIGURATION ===

```sql
SELECT
    srvname as server_name,
    srvowner::regrole as owner,
    srvoptions as connection_details
FROM pg_foreign_server
WHERE srvname = 'proj_link';
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1

	server_name name	owner regrole	connection_details text[]
1	proj_link	postgres	{host=localhost,dbname=BankingTransactionB,port=5433}

```sql
-- EVIDENCE 2: Remote Account Table Sample (5 rows)
SELECT '=== REMOTE ACCOUNT SAMPLE (5 ROWS) ===' as evidence_section;
SELECT
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	evidence_section text
1	=== REMOTE ACCOUNT SAMPLE (5 ROWS) ===

STEP2

```sql
---------- Enhanced distributed join with business metrics
SELECT
    '=== ENHANCED DISTRIBUTED JOIN WITH BUSINESS CONTEXT ===' as query_t
SELECT
    t.TransID,
    t.AccountID,
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	query_title text
1	=== ENHANCED DISTRIBUTED JOIN WITH BUSINESS CONTEXT ===

```sql
ftrelid::regclass as foreign_table_name,
srvname as server_name
_foreign_table ft
_foreign_server fs ON ft.ftserver = fs.oid
s.srvname = 'proj_link';
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	foreign_table_name regclass	server_name name
1	remote_loan	proj_link

STEP 3

```sql
-- Verify manual creation
SELECT
    'Manual Teller table creation' as status,
    CASE
        WHEN EXISTS (
            SELECT 1 FROM pg_foreign_table ft
            WHERE ft.ftrelid::regclass = 'teller'::regclass
        ) THEN '✅ SUCCESS: Teller table created manually'
        ELSE '❌ FAILED: Manual creation failed'
    END as result;
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	status text	result text
1	Manual Teller table creation	✅ SUCCESS: Teller table created manually

STEP4

```sql
    -- Test if we can access the Teller table
SELECT '=== TESTING TELLER TABLE ACCESS ===' as test;

-- Check if table exists and has data
SELECT
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	test text
1	=== TESTING TELLER TABLE ACCESS ===

```sql
- Fixed distributed join with verified table names
ELECT
   '=== DISTRIBUTED JOIN: Transaction_A ⋈ Teller@proj_link ===' as query

ELECT
   t.TransID,
   t.AccountID,
   CASE
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	query_title text
1	=== DISTRIBUTED JOIN: Transaction_A ⋈ Teller@proj_link ===

STEP5

```sql
-- COMPLETE DIAGNOSTIC SCRIPT
SELECT '=== DATABASE LINK DIAGNOSTICS ===' as section;
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	section text
1	=== DATABASE LINK DIAGNOSTICS ===

```sql
-- 1. Check database link exists
SELECT
    'Database Link Status' as check_type,
    srvname as link_name,
    array_to_string(srvoptions, ', ') as connection_details
FROM pg_foreign_server
WHERE srvname = 'proj_link';
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	check_type text	link_name name	connection_details text
1	Database Link Status	proj_link	host=localhost, dbname=BankingTransactionB, port=5433

```sql
-- 2. Check what tables are available on remote server
SELECT
    'Available Remote Tables' as check_type,
    (SELECT COUNT(*) FROM (
        SELECT table_name
        FROM information_schema.tables
        WHERE table_schema = 'public'
    ) as remote_tables) as table_count;
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	check_type text	table_count bigint
1	Available Remote Tables	5

```sql
-- 3. Try to list all foreign tables
SELECT
    'Current Foreign Tables' as check_type,
    COALESCE(
        (SELECT string_agg(ft.ftrelid::regclass::text, ', ')
         FROM pg_foreign_table ft
         JOIN pg_foreign_server fs ON ft.ftserver = fs.oid
         WHERE fs.srvname = 'proj_link'),
        'No foreign tables found'
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	check_type text	foreign_tables text
1	Current Foreign Tables	remote_loan, teller

```sql
    -- ALTERNATIVE 1: Use Transaction_B instead of Teller
SELECT
    'Alternative: Join Transaction_A with Transaction_B' as query_type;
SELECT
    a.TransID as Trans_A,
    a.AccountID as Account_A
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	query_type text
1	Alternative: Join Transaction_A with Transaction_B

A3: Serial Aggregation on Transaction_ALL

Step 1: First, Ensure Parallelism is Disabled

```sql
-- Disable parallel processing for serial execution
SET max_parallel_workers_per_gather = 0;
SET enable_parallel_hash = off;
SET enable_parallel_append = off;

-- Verify serial execution settings
SELECT
    name,
    setting,
```

Data Output Messages Notifications

Showing rows: 1 to 3 Page No: 1 of 1

	name text	setting text	status text
1	enable_parallel_append	off	Serial Mode: ✗ Disabled
2	enable_parallel_hash	off	Serial Mode: ✗ Disabled
3	max_parallel_workers_per_gather	0	Serial Mode: ✓ Enabled

Step 2: Serial Aggregation by TellerID (3-10 groups)

```sql
-------
-- SERIAL Aggregation 1: Group by TellerID
SELECT '=== SERIAL AGGREGATION BY TELLERID ===' as query_title;
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	query_title text
1	=== SERIAL AGGREGATION BY TELLERID ===

```sql
    SUM(Amount) as Total_Amount,
    ROUND(AVG(Amount), 2) as Average_Amount,
    MIN(Amount) as Min_Amount,
    MAX(Amount) as Max_Amount,
    COUNT(DISTINCT AccountID) as Unique_Accounts,
    SUM(CASE WHEN Type = 'Credit' THEN Amount ELSE 0 END) as Total_Cred
    SUM(CASE WHEN Type = 'Debit' THEN Amount ELSE 0 END) as Total_Debits
FROM Transaction_ALL
GROUP BY TellerID
ORDER BY Total_Amount DESC;
```

Data Output Messages Notifications

Showing rows: 1 to 4 Page No: 1 of 1

	tellerid character varying (50)	transaction_count bigint	total_amount numeric	average_amount numeric	min_amount numeric	max_amount numeric	unique_accounts bigint	total_credi numeric
1	T003	2	6000.00	3000.00	3000.00	3000.00	1	600
2	T001	4	4001.00	1000.25	500.50	1500.00	2	300
3	T004	2	2400.50	1200.25	1200.25	1200.25	1	240
4	T002	2	501.50	250.75	250.75	250.75	1	

Activate Windows

Step 3: Serial Aggregation by Transaction Type

```sql
-- SERIAL Aggregation 2: Group by Transaction Type
SELECT '=== SERIAL AGGREGATION BY TRANSACTION TYPE ===' as query_title;

SELECT
    Type,
    COUNT(*) as Transaction_Count,
    SUM(Amount) as Total_Amount,
    ROUND(AVG(Amount), 2) as Average_Amount,
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	query_title text
1	=== SERIAL AGGREGATION BY TRANSACTION TYPE ===

```sql
    SUM(Amount) as Total_Amount,
    ROUND(AVG(Amount), 2) as Average_Amount,
    MIN(Amount) as Min_Amount,
    MAX(Amount) as Max_Amount,
    COUNT(DISTINCT TellerID) as Unique_Tellers,
    COUNT(DISTINCT AccountID) as Unique_Accounts
FROM Transaction_ALL
GROUP BY Type
ORDER BY Total_Amount DESC;
```

Data Output Messages Notifications

Showing rows: 1 to 2 Page No: 1 of 1

	type character varying (20)	transaction_count bigint	total_amount numeric	average_amount numeric	min_amount numeric	max_amount numeric	unique_tellers bigint	unique_accou bigint
1	Credit	6	11400.50	1900.08	1200.25	3000.00	3	
2	Debit	4	1502.50	375.63	250.75	500.50	2	

Step 4: Serial Aggregation by Date Range Groups

```sql
    -- SERIAL Aggregation 3: Group by date ranges (3-5 groups)
SELECT '=== SERIAL AGGREGATION BY DATE RANGES ===' as query_title;

SELECT
    CASE
        WHEN DatePerformed BETWEEN '2024-01-10' AND '2024-01-12' THEN 'E
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	query_title text
1	=== SERIAL AGGREGATION BY DATE RANGES ===

```sql
    ROUND(AVG(Amount), 2) as Average_Amount,
    COUNT(DISTINCT TellerID) as Unique_Tellers,
    COUNT(DISTINCT AccountID) as Unique_Accounts
FROM Transaction_ALL
GROUP BY
    CASE
        WHEN DatePerformed BETWEEN '2024-01-10' AND '2024-01-12' THEN 'E
        WHEN DatePerformed BETWEEN '2024-01-13' AND '2024-01-15' THEN 'M
        WHEN DatePerformed BETWEEN '2024-01-16' AND '2024-01-19' THEN 'L
        ELSE 'Other'
    END
```

Data Output Messages Notifications

Showing rows: 1 to 2 Page No: 1 of 1

	date_period text	transaction_count bigint	total_amount numeric	average_amount numeric	unique_tellers bigint	unique_accounts bigint
1	Late Jan (16-19)	8	9903.00	1237.88	4	4
2	Mid Jan (13-15)	2	3000.00	1500.00	1	1

Step 5: Serial Aggregation by Amount Size Groups

```
ERIAL Aggregation 4: Group by transaction size categories (3-5 groups)
CT '=== SERIAL AGGREGATION BY TRANSACTION SIZE ===' as query_title;

CT
CASE
    WHEN Amount < 500 THEN 'Small (< 500)'
    WHEN Amount BETWEEN 500 AND 1500 THEN 'Medium (500-1500)'
    WHEN Amount BETWEEN 1501 AND 3000 THEN 'Large (1501-3000)'
    ELSE 'Very Large (> 3000)'
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	query_title text
1	=== SERIAL AGGREGATION BY TRANSACTION SIZE ===

```
    ROUND((COUNT(*) * 100.0 / (SELECT COUNT(*) FROM Transaction_ALL)),
FROM Transaction_ALL
GROUP BY
    CASE
        WHEN Amount < 500 THEN 'Small (< 500)'
        WHEN Amount BETWEEN 500 AND 1500 THEN 'Medium (500-1500)'
        WHEN Amount BETWEEN 1501 AND 3000 THEN 'Large (1501-3000)'
        ELSE 'Very Large (> 3000)'
    END
ORDER BY Total_Amount DESC;
```

Data Output Messages Notifications

Showing rows: 1 to 3 Page No: 1 of 1

	transaction_size text	transaction_count bigint	total_amount numeric	average_amount numeric	min_amount numeric	max_amount numeric	transaction_types bigint	percentage numeric
1	Medium (500-15...	6	6401.50	1066.92	500.50	1500.00	2	60.00
2	Large (1501-3000)	2	6000.00	3000.00	3000.00	3000.00	1	20.00
3	Small (< 500)	2	501.50	250.75	250.75	250.75	1	20.00

Step 6: Complex Serial Aggregation with Multiple Groups

```sql
-- SERIAL Aggregation 5: Multi-column grouping (3-10 groups)
SELECT '=== SERIAL AGGREGATION BY TYPE AND SIZE ===' as query_title;

SELECT
    Type,
    CASE
        WHEN Amount < 1000 THEN 'Under 1000'
        ELSE '1000 and Above'
    END as Amount_Category,
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	query_title text
1	=== SERIAL AGGREGATION BY TYPE AND SIZE ===

```sql
    COUNT(DISTINCT TellerID) as Unique_Tellers,
    COUNT(DISTINCT AccountID) as Unique_Accounts
FROM Transaction_ALL
GROUP BY Type,
    CASE
        WHEN Amount < 1000 THEN 'Under 1000'
        ELSE '1000 and Above'
    END
HAVING COUNT(*) >= 1  -- Ensure we get meaningful groups
ORDER BY Type, Total_Amount DESC;
```

Data Output Messages Notifications

Showing rows: 1 to 2 Page No: 1 of 1

	type character varying (20)	amount_category text	transaction_count bigint	total_amount numeric	average_amount numeric	unique_tellers bigint	unique_accounts bigint
1	Credit	1000 and Above	6	11400.50	1900.08	3	3
2	Debit	Under 1000	4	1502.50	375.63	2	2

Step 7: Verify Serial Execution with EXPLAIN

```sql
 Verify the queries are running in SERIAL mode with EXPLAIN
LECT '=== VERIFYING SERIAL EXECUTION WITH EXPLAIN ===' as verify_title;

PLAIN (ANALYZE, BUFFERS, FORMAT TEXT)
LECT
  TellerID,
  COUNT(*) as Transaction_Count,
  SUM(Amount) as Total_Amount
OM Transaction_ALL
OUP BY TellerID
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	verify_title text
1	=== VERIFYING SERIAL EXECUTION WITH EXPLAIN ===

```sql
EXPLAIN (ANALYZE, BUFFERS, FORMAT TEXT)
SELECT
    TellerID,
    COUNT(*) as Transaction_Count,
    SUM(Amount) as Total_Amount
FROM Transaction_ALL
GROUP BY TellerID
ORDER BY Total_Amount DESC;

-- Look for "Workers" in the plan - should show no parallel workers
```

Data Output Messages Notifications

Showing rows: 1 to 17 Page No: 1 of 1

	QUERY PLAN text
1	Sort (cost=45.44..45.94 rows=200 width=158) (actual time=0.082..0.083 rows=4.00 loops=1)
2	Sort Key: (sum(transaction_a.amount)) DESC
3	Sort Method: quicksort Memory: 25kB
4	Buffers: shared hit=1
5	-> HashAggregate (cost=35.30..37.80 rows=200 width=158) (actual time=0.049..0.051 rows=4.00 loops=1)
6	Group Key: transaction_a.tellerid

Activate Windows
Go to Settings to activate Windows.

Step 8: Row Count Validation

```sql
e get 3-10 groups/rows from our aggregations
UNT VALIDATION (3-10 GROUPS) ===' as validation_title;

ounts AS (
ID Groups' as aggregation_type, COUNT(*) as group_count
ellerID FROM Transaction_ALL GROUP BY TellerID) t
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	validation_title text
1	=== ROW COUNT VALIDATION (3-10 GROUPS) ===

Step 9: Final Evidence Query

```
NAL EVIDENCE: Serial aggregation with exactly the required output
T '=== FINAL SERIAL AGGREGATION EVIDENCE (3-10 ROWS) ===' as evidence_t

T
ellerID,
OUNT(*) as Transaction_Count,
UM(Amount) as Total_Amount,
OUND(AVG(Amount), 2) as Average_Amount,
IN(Amount) as Min_Amount,
AX(Amount) as Max_Amount,
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	evidence_title text
1	=== FINAL SERIAL AGGREGATION EVIDENCE (3-10 ROWS) ===

```
    MIN(Amount) as Min_Amount,
    MAX(Amount) as Max_Amount,
    COUNT(DISTINCT Type) as Transaction_Types_Handled
FROM Transaction_ALL
GROUP BY TellerID
HAVING COUNT(*) >= 1  -- Ensure all groups are included
ORDER BY Total_Amount DESC;

-- Show execution plan to prove it's serial
SELECT '=== EXECUTION PLAN PROVING SERIAL MODE ===' as plan_title;
EXPLAIN (ANALYZE, COSTS, VERBOSE)
```

Data Output Messages Notifications

Showing rows: 1 to 4 Page No: 1 of 1

	tellerid character varying (50)	transaction_count bigint	total_amount numeric	average_amount numeric	min_amount numeric	max_amount numeric	transaction_types_handled bigint
1	T003	2	6000.00	3000.00	3000.00	3000.00	1
2	T001	4	4001.00	1000.25	500.50	1500.00	2
3	T004	2	2400.50	1200.25	1200.25	1200.25	1
4	T002	2	501.50	250.75	250.75	250.75	1

```sql
    COUNT(DISTINCT Type) as Transaction_Types_Handled
FROM Transaction_ALL
GROUP BY TellerID
HAVING COUNT(*) >= 1  -- Ensure all groups are included
ORDER BY Total_Amount DESC;

-- Show execution plan to prove it's serial
SELECT '=== EXECUTION PLAN PROVING SERIAL MODE ===' as plan_title;
EXPLAIN (ANALYZE, COSTS, VERBOSE)
SELECT TellerID, COUNT(*), SUM(Amount)
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	plan_title text
1	=== EXECUTION PLAN PROVING SERIAL MODE ===

```sql
GROUP BY TellerID
HAVING COUNT(*) >= 1  -- Ensure all groups are included
ORDER BY Total_Amount DESC;

-- Show execution plan to prove it's serial
SELECT '=== EXECUTION PLAN PROVING SERIAL MODE ===' as plan_title;
EXPLAIN (ANALYZE, COSTS, VERBOSE)
SELECT TellerID, COUNT(*), SUM(Amount)
FROM Transaction_ALL
GROUP BY TellerID;
```

Data Output Messages Notifications

Showing rows: 1 to 16 Page No: 1 of 1

	QUERY PLAN text
1	HashAggregate (cost=35.30..37.80 rows=200 width=158) (actual time=0.066..0.069 rows=4.00 loops=1)
2	Output: transaction_a.tellerid, count(*), sum(transaction_a.amount)
3	Group Key: transaction_a.tellerid
4	Batches: 1 Memory Usage: 32kB
5	Buffers: shared hit=1
6	-> Append (cost=0.00..30.20 rows=680 width=136) (actual time=0.031..0.041 rows=10.00 loops=1)

Activate Windows
Go to Settings to activate Windows.

Step 10: Performance Comparison Baseline

```sql
------- step 10
-- Create performance baseline for serial execution
SELECT '=== SERIAL PERFORMANCE BASELINE ===' as performance_title;
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	performance_title text
1	=== SERIAL PERFORMANCE BASELINE ===

```sql
-- Time the serial aggregation
SELECT
    Type,
    COUNT(*) as Count,
    SUM(Amount) as Total
FROM Transaction_ALL
GROUP BY Type
ORDER BY Total DESC;
```

Data Output Messages Notifications

Showing rows: 1 to 2 Page No: 1 of 1

	type character varying (20)	count bigint	total numeric
1	Credit	6	11400.50
2	Debit	4	1502.50

Q3 :Parallel Aggregation with Forced Parallelism

Step 1: Enable Parallel Processing

```sql
-- Enable parallel processing with aggressive settings
SET max_parallel_workers_per_gather = 8;
SET parallel_setup_cost = 1;
SET parallel_tuple_cost = 0.001;
SET min_parallel_table_scan_size = 1;
SET min_parallel_index_scan_size = 1;
SET enable_parallel_hash = on;
SET enable_parallel_append = on;

-- Verify parallel settings
```

Data Output | Messages | Notifications

Showing rows: 1 to 5 | Page No: 1 of 1

	name text	setting text	status text
1	enable_parallel_hash	on	Parallel Mode: ✗ Disabled
2	max_parallel_workers_per_gather	8	Parallel Mode: ✓ Enabled
3	min_parallel_table_scan_size	1	Parallel Mode: ✗ Disabled
4	parallel_setup_cost	1	Parallel Mode: ✗ Disabled
5	parallel_tuple_cost	0.001	Parallel Mode: ✗ Disabled

Activate Windows

Step 2: Force Parallel Aggregation on Transaction_ALL

```sql
-- PARALLEL Aggregation 1: Force parallel execution on Transaction_ALL
SELECT '=== PARALLEL AGGREGATION BY TELLERID ===' as query_title;

SELECT
    TellerID,
    COUNT(*) as Transaction_Count,
    SUM(Amount) as Total_Amount,
    ROUND(AVG(Amount), 2) as Average_Amount,
    MIN(Amount) as Min_Amount,
```

Data Output | Messages | Notifications

Showing rows: 1 to 1 | Page No: 1 of 1

	query_title text
1	=== PARALLEL AGGREGATION BY TELLERID ===

```
    ROUND(AVG(Amount), 2) as Average_Amount,
    MIN(Amount) as Min_Amount,
    MAX(Amount) as Max_Amount,
    COUNT(DISTINCT AccountID) as Unique_Accounts,
    SUM(CASE WHEN Type = 'Credit' THEN Amount ELSE 0 END) as Total_Cred
    SUM(CASE WHEN Type = 'DeBit' THEN Amount ELSE 0 END) as Total_Debit
FROM Transaction_ALL
GROUP BY TellerID
ORDER BY Total_Amount DESC;
```

Data Output Messages Notifications

Showing rows: 1 to 4 Page No: 1 of 1

	tellerid character varying (50)	transaction_count bigint	total_amount numeric	average_amount numeric	min_amount numeric	max_amount numeric	unique_accounts bigint	total_cred numeric
1	T003	2	6000.00	3000.00	3000.00	3000.00	1	60
2	T001	4	4001.00	1000.25	500.50	1500.00	2	30
3	T004	2	2400.50	1200.25	1200.25	1200.25	1	24
4	T002	2	501.50	250.75	250.75	250.75	1	

Activate Windows

Step 3: Verify Parallel Execution with EXPLAIN

```
-- Verify parallel execution is being used
SELECT '=== VERIFYING PARALLEL EXECUTION WITH EXPLAIN ===' as verify_t

EXPLAIN (ANALYSE, BUFFERS, COSTS, VERBOSE, FORMAT TEXT)
SELECT
    TellerID,
    COUNT(*) as Transaction_Count,
    SUM(Amount) as Total_Amount
FROM Transaction_ALL
GROUP BY TellerID
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	verify_title text
1	=== VERIFYING PARALLEL EXECUTION WITH EXPLAIN ===

```sql
EXPLAIN (ANALYSE, BUFFERS, COSTS, VERBOSE, FORMAT TEXT)
SELECT
    TellerID,
    COUNT(*) as Transaction_Count,
    SUM(Amount) as Total_Amount
FROM Transaction_ALL
GROUP BY TellerID
ORDER BY Total_Amount DESC;
```

Data Output Messages Notifications

Showing rows: 1 to 29 Page No: 1 of 1

	QUERY PLAN text
1	Sort (cost=40.22..40.72 rows=200 width=158) (actual time=71.129..75.556 rows=4.00 loops=1)
2	Output: transaction_a.tellerid, (count(*)), (sum(transaction_a.amount))
3	Sort Key: (sum(transaction_a.amount)) DESC
4	Sort Method: quicksort Memory: 25kB
5	Buffers: shared hit=1
6	-> HashAggregate (cost=30.07..32.57 rows=200 width=158) (actual time=71.111..75.540 rows=4.00 loops=1)

Activate Windows
Go to Settings to activate Windows.

Step 4: Alternative Method - Use Large Table Simulation

```sql
- Add primary key for the temporary table
.TER TABLE transaction_large ADD PRIMARY KEY (TransID);

- Analyze the table to update statistics
IALYZE transaction_large;

- PARALLEL Aggregation on larger dataset
ELECT '=== PARALLEL AGGREGATION ON LARGER DATASET ===' as query_title;
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	query_title text
1	=== PARALLEL AGGREGATION ON LARGER DATASET ===

```sql
    COUNT(*) as Transaction_Count,
    SUM(Amount) as Total_Amount,
    ROUND(AVG(Amount), 2) as Average_Amount,
    MIN(Amount) as Min_Amount,
    MAX(Amount) as Max_Amount
FROM transaction_large
GROUP BY TellerID
ORDER BY Total_Amount DESC
LIMIT 8;  -- Limit to 3-10 rows
```

Data Output Messages Notifications

Showing rows: 1 to 4 Page No: 1 of 1

	tellerid character varying (50)	transaction_count bigint	total_amount numeric	average_amount numeric	min_amount numeric	max_amount numeric
1	T003	2000	7007260.00	3503.63	3000.00	4000.00
2	T001	4000	6014218.00	1503.55	501.50	2500.00
3	T004	2000	3430636.00	1715.32	1200.25	2200.25
4	T002	2000	1484353.00	742.18	250.75	1250.75

Step 5: PostgreSQL-Style Parallel Hints

```sql

ostgreSQL, we can use SET to force parallelism for the session
'=== FORCING PARALLELISM WITH POSTGRESQL SETTINGS ===' as method_title;

od 1: Use pg_hint_plan extension if available
t check if pg_hint_plan is installed
EXISTS(
ECT 1 FROM pg_extension WHERE extname = 'pg_hint_plan'
s_hint_plan;
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	method_title text
1	=== FORCING PARALLELISM WITH POSTGRESQL SETTINGS ===

```
1253    SELECT 1 FROM pg_extension WHERE extname = 'pg_hint_plan'
1254  as has_hint_plan;
1255
1256  - If pg_hint_plan is available, use it like this:
1257  *
1258  OAD 'pg_hint_plan';
1259  ET pg_hint_plan.enable_hint = on;
1260
1261  - This would be the equivalent of Oracle's PARALLEL hint
1262  ELECT /*+ Parallel(transaction_large 8) */
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	has_hint_plan boolean
1	false

```
1297
1298  rison: Serial vs Parallel
1299  NCE COMPARISON: SERIAL VS PARALLEL ===' as comparison_title;
1300
1301  timing
1302  kers_per_gather = 0;
1303
1304  NT(*), SUM(Amount)
1305  ge
1306
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	comparison_title text
1	=== PERFORMANCE COMPARISON: SERIAL VS PARALLEL ===

```
1330            ELSE 'Large'
1331        END as Size_Category,
1332        COUNT(*) as Count,
1333        SUM(Amount) as Total
1334    FROM transaction_large
1335    GROUP BY TellerID, Type,
1336        CASE
1337            WHEN Amount < 500 THEN 'Small'
1338            WHEN Amount BETWEEN 500 AND 1500 THEN 'Medium'
1339            ELSE 'Large'
1340        END
```

Data Output Messages Notifications

Showing rows: 1 to 7 Page No: 1 of 1

	tellerid character varying (50)	type character varying (20)	size_category text	count bigint	total numeric	average numeric
1	T003	Credit	Large	2000	7007260.00	3503.63
2	T001	Credit	Large	1999	4000627.00	2001.31
3	T004	Credit	Large	1444	2680991.00	1856.64
4	T001	Debit	Medium	2000	2012091.00	1006.05
5	T002	Debit	Medium	1482	1288991.50	869.76
6	T004	Credit	Medium	556	749645.00	1348.28

Activate Windows
Go to Settings to activate Windows.

Q3.PostgreSQL Equivalent: Execution Plans and Statistics

Step 1: Enable Detailed Statistics and Timing

```
  name,
  setting,
  'Statistics: ' ||
  CASE
      WHEN name = 'track_io_timing' AND setting = 'on' THEN '✅ Enablec
      WHEN name = 'track_activities' AND setting = 'on' THEN '✅ Enable
      ELSE '❌ Disabled'
  END as status
OM pg_settings
ERE name IN ('track_io_timing', 'track_activities', 'track_functions');
```

Data Output Messages Notifications

Showing rows: 1 to 3 Page No: 1 of 1

	name text	setting text	status text
1	track_activities	on	Statistics: ✅ Enabled
2	track_functio...	none	Statistics: ❌ Disabled
3	track_io_timi...	off	Statistics: ❌ Disabled

Step 2: Capture Serial Execution Plan with Statistics

```
-- SERIAL EXECUTION: Capture detailed plan and statistics
SELECT '=== SERIAL EXECUTION PLAN & STATISTICS ===' as title;

-- Disable parallelism for serial execution
SET max_parallel_workers_per_gather = 0;
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	title text
1	=== SERIAL EXECUTION PLAN & STATISTICS ===

```sql
-- Execute with detailed EXPLAIN ANALYZE (PostgreSQL's DBMS_XPLAN equi
EXPLAIN (ANALYZE, BUFFERS, COSTS, VERBOSE, FORMAT YAML)
SELECT
    TellerID,
    COUNT(*) as Transaction_Count,
    SUM(Amount) as Total_Amount,
    ROUND(AVG(Amount), 2) as Average_Amount
FROM Transaction_ALL
GROUP BY TellerID
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	QUERY PLAN text
1	- Plan: Node Type: "Sort" Parallel Aware: false Async Capable: false Startup Cost: 46.44 Total Cost: 46.94 Plan Rows: 200 Plan Width: 190 Actual

Comprehensive Statistics Collection

```sql
-------
Comprehensive statistics collection (PostgreSQL's AUTOTRACE equivalent)
ECT '=== COMPREHENSIVE QUERY STATISTICS (AUTOTRACE EQUIVALENT) ===' as

Get baseline statistics before query
ECT
 'Pre-Query Statistics' as stats_type,
 schemaname,
 relname as table_name,
 seq_scan,
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	title text
1	=== COMPREHENSIVE QUERY STATISTICS (AUTOTRACE EQUIVALENT) ===

```sql
    schemaname,
    relname as table_name,
    seq_scan,
    seq_tup_read,
    idx_scan,
    n_tup_ins,
    n_tup_upd,
    n_tup_del
FROM pg_stat_user_tables
WHERE relname LIKE 'transaction%';
```

Data Output Messages Notifications

Showing rows: 1 to 2 Page No: 1 of 1

	stats_type text	schemaname name	table_name name	seq_scan bigint	seq_tup_read bigint	idx_scan bigint	n_tup_ins bigint	n_tup_upd bigint	n_tup_del bigint
1	Pre-Query Statistics	public	transaction_a	91	810	0	10	0	0
2	Pre-Query Statistics	public	transaction_b	94	0	0	0	0	0

```sql
-- Execute query with full statistics
EXPLAIN (ANALYZE, BUFFERS, COSTS, TIMING, SUMMARY, FORMAT TEXT)
SELECT
    Type,
    COUNT(*) as Count,
    SUM(Amount) as Total,
    ROUND(AVG(Amount), 2) as Average
FROM Transaction_ALL
GROUP BY Type
ORDER BY Total DESC;
```

Data Output | Messages | Notifications

Showing rows: 1 to 17 | Page No: 1 of 1

	QUERY PLAN text
1	Sort (cost=46.44..46.94 rows=200 width=130) (actual time=0.138..0.139 rows=2.00 loops=1)
2	Sort Key: (sum(transaction_a.amount)) DESC
3	Sort Method: quicksort Memory: 25kB
4	Buffers: shared hit=1
5	-> HashAggregate (cost=35.30..38.80 rows=200 width=130) (actual time=0.123..0.125 rows=2.00 loops=1)
6	Group Key: transaction_a.type

Activate Windows
Go to Settings to activate Windows.

```sql
    schemaname,
    relname as table_name,
    seq_scan,
    seq_tup_read,
    idx_scan,
    n_tup_ins,
    n_tup_upd,
    n_tup_del
FROM pg_stat_user_tables
WHERE relname LIKE 'transaction%';
```

Data Output | Messages | Notifications

Showing rows: 1 to 2 | Page No: 1 of 1

	stats_type text	schemaname name	table_name name	seq_scan bigint	seq_tup_read bigint	idx_scan bigint	n_tup_ins bigint	n_tup_upd bigint	n_tup_del bigint
1	Post-Query Statistics	public	transaction_a	92	820	0	10	0	0
2	Post-Query Statistics	public	transaction_b	95	0	0	0	0	0

A4: Two-Phase Commit & Recovery

Step 1: Create Required Tables

Query | Query History | Scratch Pad

```sql
SELECT audit_id, transaction_id, amount, description, status, created_a
FROM local_transaction_audit
ORDER BY created_at DESC
LIMIT 3;

SELECT 'Remote Payment Records:' as table_name;
SELECT paymentid, loanid, amount, paymentdate, mode
FROM payment
ORDER BY paymentdate DESC
LIMIT 3;
```

Data Output | Messages | Notifications

```
WARNING:  there is no transaction in progress
COMMIT

Query returned successfully in 62 msec.
```

Query | Query History | Scratch Pad

```sql
        ELSE 'In Progress'
    END as recovery_status,
    local_data,
    remote_data
FROM two_phase_commit_log
ORDER BY created_at DESC;

-- Check recovery status
SELECT '=== TWO-PHASE COMMIT RECOVERY STATUS ===' as status_check;
SELECT * FROM two_phase_recovery_status;
```

Data Output | Messages | Notifications

Showing rows: 1 to 1 | Page No: 1 of 1

	status_check text
1	=== TWO-PHASE COMMIT RECOVERY STATUS ===

Q2. Create Failure Simulation Setup

```sql
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Verify current row counts to ensure we're within budget
SELECT '=== CURRENT ROW COUNT VERIFICATION (Pre-Failure Test) ===' as v
SELECT
    'local_transaction_audit' as table_name,
    COUNT(*) as row_count
FROM local_transaction_audit
UNION ALL
```

Data Output | Messages | Notifications

Showing rows: 1 to 1 | Page No: 1 of 1

	verification text
1	=== CURRENT ROW COUNT VERIFICATION (Pre-Failure Test) ===

```
----------
-- Check if any prepared transactions exist (true in-doubt state)
SELECT '=== CHECKING FOR IN-DOUBT TRANSACTIONS ===' as check;
```

Data Output | Messages | Notifications

Showing rows: 1 to 1 | Page No: 1 of 1

	check text
1	=== CHECKING FOR IN-DOUBT TRANSACTIONS ===

```
SELECT
    gid as transaction_id,
    prepared as prepared_at,
    owner as transaction_owner,
    database
FROM pg_prepared_xacts;

-- If any exist, show how to resolve them
```

Data Output | Messages | Notifications

transaction_id text	prepared_at timestamp with time zone	transaction_owner name	database name

```
FROM pg_prepared_xacts;

-- If any exist, show how to resolve them
SELECT
    CASE
        WHEN EXISTS (SELECT 1 FROM pg_prepared_xacts) THEN
            'IN-DOUBT TRANSACTIONS FOUND: Use COMMIT PREPARED or ROLLBA
        ELSE
            'No in-doubt transactions found'
    END as in_doubt_status;
```

Data Output | Messages | Notifications

Showing rows: 1 to 1 | Page No: 1 of 1

	in_doubt_status text
1	No in-doubt transactions found

Q3 .Check PostgreSQL Equivalent of DBA_2PC_PENDING

```
);

-- Verify current row counts to ensure we're within budget
SELECT '=== CURRENT ROW COUNT VERIFICATION (Pre-Failure Test) ===' as v
SELECT
    'local_transaction_audit' as table_name,
    COUNT(*) as row_count
FROM local_transaction_audit
UNION ALL
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	verification text
1	=== CURRENT ROW COUNT VERIFICATION (Pre-Failure Test) ===

```

        -- Automatic rollback of local row
        DELETE FROM local_transaction_audit
        WHERE transaction_id = (SELECT MAX(transaction_id) FROM local_tr
          AND description LIKE '%Network Failure Simulation%';

        UPDATE failure_simulation_log
        SET status = 'Rolled Back',
            error_message = 'Remote insert failed: ' || SQLERRM
        WHERE simulation_id = sim_id;
```

Data Output Messages Notifications

```
WARNING:  there is no transaction in progress
COMMIT

Query returned successfully in 41 msec.
```

```
    'failure_simulation_log' as table_name,
    COUNT(*) as row_count,
    'N/A - Log table excluded from budget' as budget_status
FROM failure_simulation_log;

-- Show only committed transactions (should be within budget)
SELECT '4. COMMITTED TRANSACTIONS ONLY:' as check_type;
SELECT
    'local_transaction_audit' as source,
    transaction_id,
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	check_type text
1	4. COMMITTED TRANSACTIONS ONLY:

```sql
-- Current row counts across all tables
SELECT '3. CURRENT ROW COUNTS (Post-Failure Tests):' as check_type;
SELECT
    'local_transaction_audit' as table_name,
    COUNT(*) as row_count,
    CASE
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	check_type text
1	3. CURRENT ROW COUNTS (Post-Failure Tests):

```sql
----------

-- Comprehensive verification of rollbacks and row budget
SELECT '=== COMPREHENSIVE ROLLBACK VERIFICATION ===' as verification;

-- Check failure simulation results
SELECT '1. FAILURE SIMULATION RESULTS:' as check_type;
SELECT
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	verification text
1	=== COMPREHENSIVE ROLLBACK VERIFICATION ===

```sql
-- Check failure simulation results
SELECT '1. FAILURE SIMULATION RESULTS:' as check_type;
SELECT
    simulation_type,
    status,
    local_row_created,
    remote_row_created,
    error_message,
    created_at
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	check_type text
1	1. FAILURE SIMULATION RESULTS:

```sql
SELECT
    simulation_type,
    status,
    local_row_created,
    remote_row_created,
    error_message,
    created_at
FROM failure_simulation_log
ORDER BY created_at DESC;
```

Data Output Messages Notifications

simulation_type character varying (50)	status character varying (20)	local_row_created boolean	remote_row_created boolean	error_message text	created_at timestamp without time zone

```sql

-- Verify no orphaned local rows from failed transactions
SELECT '2. ORPHANED LOCAL ROWS CHECK:' as check_type;
SELECT
    COUNT(*) as orphaned_rows,
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	check_type text
1	2. ORPHANED LOCAL ROWS CHECK:

```sql
SELECT
    COUNT(*) as orphaned_rows,
    'All failure simulation rows should be rolled back' as note
FROM local_transaction_audit
WHERE description LIKE '%Simulation%'
  AND status != 'Committed';

-- Current row counts across all tables
SELECT '3. CURRENT ROW COUNTS (Post-Failure Tests):' as check_type;
SELECT
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1

	orphaned_rows bigint	note text
1	0	All failure simulation rows should be rolled back

```sql
----------
-- Check if any prepared transactions exist (true in-doubt state)
SELECT '=== CHECKING FOR IN-DOUBT TRANSACTIONS ===' as check;

SELECT
    gid as transaction_id,
    prepared as prepared_at,
    owner as transaction_owner,
    database
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	check text
1	=== CHECKING FOR IN-DOUBT TRANSACTIONS ===

```sql
FROM pg_prepared_xacts;

-- If any exist, show how to resolve them
SELECT
    CASE
        WHEN EXISTS (SELECT 1 FROM pg_prepared_xacts) THEN
            'IN-DOUBT TRANSACTIONS FOUND: Use COMMIT PREPARED or ROLLBA
        ELSE
            'No in-doubt transactions found'
    END as in_doubt_status;
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	in_doubt_status text
1	No in-doubt transactions found

Q4.Clean Two-Phase Commit PL/pgSQL Block

```sql
-- CHECK FOR PENDING TRANSACTIONS (BEFORE ANY ACTION)
SELECT '=== PENDING TRANSACTIONS CHECK (BEFORE) ===' as status;

-- Method 1: Prepared transactions (true in-doubt transactions)
SELECT
    'Prepared Transactions (DBA_2PC_PENDING equivalent)' as query_type,
    COUNT(*) as pending_count,
    COALESCE(string_agg(gid, ', '), 'None') as transaction_ids,
    CASE
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	status text
1	=== PENDING TRANSACTIONS CHECK (BEFORE) ===

Query | Query History | Scratch Pad

```
    END as status
FROM pg_prepared_xacts;

-- Method 2: Long-running transactions that might appear as pending
SELECT
    'Long-running Transactions' as query_type,
    COUNT(*) as long_running_count,
    string_agg(pid::text, ', ') as process_ids,
    CASE
        WHEN COUNT(*) = 0 THEN '✅ CLEAN - No long-running transaction
```

Data Output | Messages | Notifications

Showing rows: 1 to 1 | Page No: 1 of 1

	query_type text	pending_count bigint	transaction_ids text	status text
1	Prepared Transactions (DBA_2PC_PENDING equivalent)	0	None	✅ CLEAN - No pending transactions

```
    END as status
FROM pg_stat_activity
WHERE state IN ('idle in transaction', 'active')
  AND now() - state_change > interval '1 minute'
  AND datname = current_database();

-- Method 3: Check for transaction locks
SELECT
    'Transaction Locks' as query_type,
```

Data Output | Messages | Notifications

Showing rows: 1 to 1 | Page No: 1 of 1

	query_type text	long_running_count bigint	process_ids text	status text
1	Long-running Transactions	0	[null]	✅ CLEAN - No long-running transactions

```
    'Transaction Locks' as query_type,
    COUNT(*) as lock_count,
    COUNT(CASE WHEN NOT granted THEN 1 END) as waiting_locks,
    CASE
        WHEN COUNT(CASE WHEN NOT granted THEN 1 END) = 0 THEN '✅ CLEA
        ELSE '❌ BLOCKED - ' || COUNT(CASE WHEN NOT granted THEN 1 END
    END as status
FROM pg_locks
WHERE locktype IN ('transactionid', 'virtualxid');
```

Data Output | Messages | Notifications

Showing rows: 1 to 1 | Page No: 1 of 1

	query_type text	lock_count bigint	waiting_locks bigint	status text
1	Transaction Locks	1	0	✅ CLEAN - No waiting locks

.A5: Distributed Lock Conflict & Diagnosis

Q1: Session 1 - Node_A (Blocker Session)

```sql
-- SESSION 1: Node_A (Blocker) - Keep this transaction OPEN
BEGIN;

-- Update a row in local_transaction_audit (we'll use an existing row)
UPDATE local_transaction_audit
SET amount = amount + 100,
    description = description || ' - Locked by Session 1'
WHERE transaction_id = (
    SELECT transaction_id FROM local_transaction_audit
    WHERE status = 'Committed'
```

Data Output Messages Notifications

transaction_id integer	amount numeric (15,2)	description text

Q2: Session 2 - Node_B via proj_link (Waiter Session)

```sql
-------Q2
-- SESSION 2: Node_B via proj_link (Waiter) - Run this in a separate co
BEGIN;

-- Try to update the same logical row via the foreign table
-- This will BLOCK waiting for Session 1 to commit or rollback
UPDATE local_transaction_audit
SET amount = amount - 50,
    description = description || ' - Modified by Session 2'
WHERE transaction_id = (
```

Data Output Messages Notifications

transaction_id integer	amount numeric (15,2)	description text

Q3: Session 3 - Node_A (Diagnostics Session)

```
--------Q3

-- SESSION 3: Node_A (Diagnostics) - Run this in a third connection
SELECT '=== DISTRIBUTED LOCK CONFLICT DIAGNOSTICS ===' as title;

-- 1. Identify blocking and waiting sessions (PostgreSQL equivalent of
SELECT '1. BLOCKER/WAITER SESSION ANALYSIS:' as analysis;
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	title text
1	=== DISTRIBUTED LOCK CONFLICT DIAGNOSTICS ===

```
-- 1. Identify blocking and waiting sessions (PostgreSQL equivalent of
SELECT '1. BLOCKER/WAITER SESSION ANALYSIS:' as analysis;

SELECT
    blocked_locks.pid AS blocked_pid,
    blocked_activity.usename AS blocked_user,
    blocking_locks.pid AS blocking_pid,
    blocking_activity.usename AS blocking_user,
    blocked_activity.query AS blocked_statement,
    blocking_activity.query AS current_statement_in_blocking_process,
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	analysis text
1	1. BLOCKER/WAITER SESSION ANALYSIS:

Query Query History Scratch Pad

```
    AND blocking_locks.page IS NOT DISTINCT FROM blocked_locks.page
    AND blocking_locks.tuple IS NOT DISTINCT FROM blocked_locks.tuple
    AND blocking_locks.virtualxid IS NOT DISTINCT FROM blocked_locks.vi
    AND blocking_locks.transactionid IS NOT DISTINCT FROM blocked_locks
    AND blocking_locks.classid IS NOT DISTINCT FROM blocked_locks.class
    AND blocking_locks.objid IS NOT DISTINCT FROM blocked_locks.objid
    AND blocking_locks.objsubid IS NOT DISTINCT FROM blocked_locks.objs
    AND blocking_locks.pid != blocked_locks.pid
JOIN pg_catalog.pg_stat_activity blocking_activity ON blocking_activity
WHERE NOT blocked_locks.GRANTED;
```

Data Output Messages Notifications

	blocked_pid integer	blocked_user name	blocking_pid integer	blocking_user name	blocked_statement text	current_statement_in_blocking_process text	blocked_app text

```sql
-- 2. Detailed lock information (PostgreSQL equivalent of V$LOCK)
SELECT '2. DETAILED LOCK INFORMATION:' as info;

SELECT
    lock.locktype,
    lock.mode,
    lock.granted,
    lock.pid,
    usename as username,
    datname as database name,
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	info text
1	2. DETAILED LOCK INFORMATION:

```sql
    objid,
    objsubid
FROM pg_locks lock
JOIN pg_stat_activity activity ON lock.pid = activity.pid
WHERE lock.pid IN (
    SELECT pid FROM pg_locks
    WHERE relation = 'local_transaction_audit'::regclass
       OR virtualxid IS NOT NULL
)
ORDER BY lock.pid, lock.granted;
```

Data Output Messages Notifications

Showing rows: 1 to 45 Page No: 1 of 1

	locktype text	mode text	granted boolean	pid integer	username name	database_name name	relation_name regclass	page integer
1	relation	AccessShareLock	true	14280	postgres	SECTIONB	pg_attrdef_adrelid_adnum_index	[nu
2	relation	AccessShareLock	true	14280	postgres	SECTIONB	pg_sequence	[nu
3	relation	AccessShareLock	true	14280	postgres	SECTIONB	pg_depend	[nu
4	relation	AccessShareLock	true	14280	postgres	SECTIONB	pg_attrdef	[nu
5	relation	AccessShareLock	true	14280	postgres	SECTIONB	pg_constraint_conname_nsp_index	[nu

Activate Windows

```sql
-- 3. Current activity showing waiting queries
SELECT '3. CURRENT ACTIVITY WITH WAITING QUERIES:' as activity;

SELECT
    pid,
    usename,
    application_name,
    client_addr,
    state,
    wait_event_type,
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	activity text
1	3. CURRENT ACTIVITY WITH WAITING QUERIES:

```sql
    state,
    wait_event_type,
    wait_event,
    now() - query_start as query_duration,
    query
FROM pg_stat_activity
WHERE state != 'idle'
  AND query IS NOT NULL
  AND datname = current_database()
ORDER BY query_start;
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	pid integer	usename name	application_name text	client_addr inet	state text	wait_event_type text	wait_event text	query_duration interval	query text
1	14280	postgres	pgAdmin 4 - CONN:8982095	::1	active	[null]	[null]	-00:10:56.52374	SELECT blocke

```sql
SELECT
    'local_transaction_audit' as table_name,
    COUNT(*) as total_locks,
    COUNT(CASE WHEN granted THEN 1 END) as granted_locks,
    COUNT(CASE WHEN NOT granted THEN 1 END) as waiting_locks,
    string_agg(
        CASE WHEN granted THEN 'PID ' || pid || ': ' || mode
             ELSE 'PID ' || pid || ': ' || mode || ' (WAITING)'
        END, ', '
    ) as lock_details
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	table_locks text
1	4. TABLE-SPECIFIC LOCK CONFLICT:

```sql
    COUNT(CASE WHEN granted THEN 1 END) as granted_locks,
    COUNT(CASE WHEN NOT granted THEN 1 END) as waiting_locks,
    string_agg(
        CASE WHEN granted THEN 'PID ' || pid || ': ' || mode
             ELSE 'PID ' || pid || ': ' || mode || ' (WAITING)'
        END, ', '
    ) as lock_details
FROM pg_locks
WHERE relation = 'local_transaction_audit'::regclass;
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	table_name text	total_locks bigint	granted_locks bigint	waiting_locks bigint	lock_details text
1	local_transaction_audit	2	2	0	PID 14280: AccessShareLock, PID 14280: RowExclusiveLock

```
            ELSE 'PID ' || pid || ': ' || mode || ' (WAITING)'
        END, ', '
    ) as lock_details
FROM pg_locks
WHERE relation = 'local_transaction_audit'::regclass;

-- 5. Timestamp for evidence
SELECT '5. CURRENT TIMESTAMP FOR EVIDENCE:' as timestamp_info;
SELECT now() as current_time;
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	timestamp_info text
1	5. CURRENT TIMESTAMP FOR EVIDENCE:

```
-- 5. Timestamp for evidence
SELECT '5. CURRENT TIMESTAMP FOR EVIDENCE:' as timestamp_info;
SELECT now() as current_time;
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	current_time timestamp with time zone
1	2025-10-29 17:23:31.93345-07

Q 4: Session 1 - Release the Lock

```
    -- SESSION 1: Node_A (Back in the original blocking session)
-- Now release the lock by committing
COMMIT;

SELECT 'Session 1: Lock released via COMMIT' as status;
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	status text
1	Session 1: Lock released via COMMIT

```sql
--------Q4 Step 4: Session 1 - Release the Lock

    -- SESSION 1: Node_A (Back in the original blocking session)
-- Now release the lock by committing
COMMIT;

SELECT 'Session 1: Lock released via COMMIT' as status;
SELECT 'Session 2 should now complete its update' as expected_result;
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	expected_result text
1	Session 2 should now complete its update

```sql
-- SESSION 2: Node_B (Back in the waiting session)
-- After Session 1 commits, this should now complete
SELECT 'Session 2: Update completed after lock release' as status;

-- Verify the update was applied
SELECT transaction_id, amount, description
FROM local_transaction_audit
WHERE transaction_id = (
    SELECT transaction_id FROM local_transaction_audit
    WHERE description LIKE '%Modified by Session 2%'
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	status text
1	Session 2: Update completed after lock release

B6: Declarative Rules Hardening

Q 1: Add NOT NULL and CHECK Constraints

```sql
-- ADD CONSTRAINTS TO PAYMENT TABLE
SELECT '=== ADDING CONSTRAINTS TO PAYMENT TABLE ===' as step;

-- Check current Payment table structure
SELECT
    column_name,
    data_type,
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	step text
1	=== ADDING CONSTRAINTS TO PAYMENT TABLE ===

```sql
-- Check current Payment table structure
SELECT
    column_name,
    data_type,
    is_nullable,
    column_default
FROM information_schema.columns
WHERE table_name = 'payment'
ORDER BY ordinal_position;
```

Data Output Messages Notifications

	column_name name	data_type character varying	is_nullable character varying (3)	column_default character varying

Q2: Test INSERTs with Proper Error Handling

```sql
-- COMPREHENSIVE TEST SCRIPT WITH ERROR HANDLING
SELECT '=== TESTING CONSTRAINTS WITH VALIDATION INSERTS ===' as step;

-- Get current row counts to ensure we stay within budget
SELECT
    'Current Row Counts (Pre-Test):' as status
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	step text
1	=== TESTING CONSTRAINTS WITH VALIDATION INSERTS ===

```sql
-- Check current Payment table structure
SELECT
    column_name,
    data_type,
    is_nullable,
    column_default
FROM information_schema.columns
WHERE table_name = 'payment'
ORDER BY ordinal_position;
```

Data Output Messages Notifications

	column_name name	data_type character varying	is_nullable character varying (3)	column_default character varying

Query Query History

Scratch Pad

```sql
    SELECT 'Starting Payment tests. Current payment rows:', COUNT(*) FI

    -- TEST 1: PASSING INSERT - Valid payment
    INSERT INTO payment (loanid, amount, paymentdate, mode)
    VALUES (1, 500.00, '2024-02-28', 'Transfer')
    RETURNING 'PASSING Payment Test 1 - ID: ' || paymentid as test_resu

    -- TEST 2: PASSING INSERT - Another valid payment
    INSERT INTO payment (loanid, amount, paymentdate, mode)
    VALUES (2, 750.00, '2024-03-01', 'Cash')
```

Data Output Messages Notifications

```
WARNING:  there is no transaction in progress
COMMIT

Query returned successfully in 102 msec.
```

Q 3: Additional Constraint Tests

```
----------Q3

-- ADDITIONAL CONSTRAINT VALIDATION TESTS
SELECT '=== ADDITIONAL CONSTRAINT VALIDATION ===' as additional_tests;
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	additional_tests text
1	=== ADDITIONAL CONSTRAINT VALIDATION ===

```
ILING INSERT - Invalid status for Loan
ional Test 1: Invalid Loan Status (Should FAIL)' as test_case;

TO loan (customerid, amount, interestrate, startdate, enddate, status)
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	test_case text
1	Additional Test 1: Invalid Loan Status (Should FAIL)

```
        INSERT INTO loan (customerid, amount, interestrate, startdate,
        VALUES (5, 8000.00, 6.5, '2024-02-01', '2024-12-01', 'InvalidSt
        RAISE EXCEPTION '❌ TEST FAILED: Invalid status should have bee
    EXCEPTION
        WHEN check_violation THEN
            RAISE NOTICE '✅ EXPECTED ERROR: %', SQLERRM;
        WHEN OTHERS THEN
            RAISE NOTICE '❌ UNEXPECTED ERROR: %', SQLERRM;
    END;
```

Data Output Messages Notifications

```
ROLLBACK

Query returned successfully in 72 msec.
```

```
 6: FAILING INSERT - Invalid payment mode
'Additional Test 2: Invalid Payment Mode (Should FAIL)' as test_case;

ERT INTO payment (loanid, amount, paymentdate, mode)
JES (1, 400.00, '2024-02-20', 'InvalidMode');
SE EXCEPTION '❌ TEST FAILED: Invalid payment mode should have been rej
ON
N check_violation THEN
```

Data Output | Messages | Notifications

Showing rows: 1 to 1 | Page No: 1 of 1

	test_case text
1	Additional Test 2: Invalid Payment Mode (Should FAIL)

```
 FAILING INSERT - Date order violation for Loan
litional Test 3: Invalid Date Order (Should FAIL)' as test_case;

 INTO loan (customerid, amount, interestrate, startdate, enddate, status
 (3, 9000.00, 7.0, '2024-12-01', '2024-02-01', 'Active');
XCEPTION '❌ TEST FAILED: End date before start date should have been r

heck_violation THEN
```

Data Output | Messages | Notifications

Showing rows: 1 to 1 | Page No: 1 of 1

	test_case text
1	Additional Test 3: Invalid Date Order (Should FAIL)

```
-- Test 8: FAILING INSERT - NULL value violation
SELECT 'Additional Test 4: NULL Amount (Should FAIL)' as test_case;
BEGIN
    INSERT INTO payment (loanid, amount, paymentdate, mode)
    VALUES (2, NULL, '2024-02-25', 'Transfer');
    RAISE EXCEPTION '❌ TEST FAILED: NULL amount should have been reje
EXCEPTION
```

Data Output | Messages | Notifications

Showing rows: 1 to 1 | Page No: 1 of 1

	test_case text
1	Additional Test 4: NULL Amount (Should FAIL)

B7: E-C-A Trigger for Denormalized Totals

Q1: Create Loan_AUDIT Table

```sql
CREATE TABLE Loan_AUDIT (
    audit_id SERIAL PRIMARY KEY,
    bef_total DECIMAL(15,2),
    aft_total DECIMAL(15,2),
    changed_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    key_col VARCHAR(64),
    operation_type VARCHAR(10),
    rows_affected INTEGER
);
select *from Loan_AUDIT;
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	status text
1	Loan_AUDIT table created successfully

```sql
);
select *from Loan_AUDIT;

-- Verify table creation
SELECT 'Loan_AUDIT table created successfully' as status;
SELECT column_name, data_type, is_nullable
FROM information_schema.columns
WHERE table_name = 'loan_audit'
ORDER BY ordinal_position;
```

Data Output Messages Notifications

Showing rows: 1 to 7 Page No: 1 of 1

	column_name name	data_type character varying	is_nullable character varying (3)
1	audit_id	integer	NO
2	bef_total	numeric	YES
3	aft_total	numeric	YES
4	changed_at	timestamp without time zone	YES
5	key_col	character varying	YES
6	operation_type	character varying	YES

Activate Windows
Go to Settings to activate Windows.

```sql
-- ADD DENORMALIZED TOTAL COLUMN TO LOAN TABLE
ALTER TABLE Loan ADD COLUMN total_payments DECIMAL(15,2) DEFAULT 0.00;

-- Initialize existing loans with their current payment totals
UPDATE Loan
SET total_payments = (
    SELECT COALESCE(SUM(amount), 0.00)
    FROM Payment
    WHERE Payment.loanid = Loan.loanid
);
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	status text
1	Loan table updated with denormalized total_payments column

Q2: Implement Statement-Level Trigger

```sql
EXECUTE FUNCTION update_loan_totals();

-- Verify trigger creation
SELECT 'Trigger created successfully' as status;
SELECT
    trigger_name,
    event_manipulation,
    action_statement,
    action_timing
FROM information_schema.triggers
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	status text
1	Trigger created successfully

```sql
-- Verify trigger creation
SELECT 'Trigger created successfully' as status;
SELECT
    trigger_name,
    event_manipulation,
    action_statement,
    action_timing
FROM information_schema.triggers
WHERE event_object_table = 'payment';
```

Data Output Messages Notifications

trigger_name name	event_manipulation character varying	action_statement character varying	action_timing character varying

Q3 : Execute Mixed DML Script (Affecting 4 Rows Total)

```sql
-- MIXED DML SCRIPT (4 ROWS TOTAL)
SELECT '=== EXECUTING MIXED DML SCRIPT (4 ROWS) ===' as script_start;

-- Get initial state for verification
SELECT 'Initial state - Loan totals:' as initial_state;
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	script_start text
1	=== EXECUTING MIXED DML SCRIPT (4 ROWS) ===

```sql
-- Get initial state for verification
SELECT 'Initial state - Loan totals:' as initial_state;
SELECT loanid, amount as loan_amount, total_payments, status FROM Loan

SELECT 'Initial state - Payment counts:' as payment_state;
SELECT loanid, COUNT(*) as payment_count, SUM(amount) as total_payments
FROM Payment
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	initial_state text
1	Initial state - Loan totals:

```sql
-- Get initial state for verification
SELECT 'Initial state - Loan totals:' as initial_state;
SELECT loanid, amount as loan_amount, total_payments, status FROM Loan C

SELECT 'Initial state - Payment counts:' as payment_state;
SELECT loanid, COUNT(*) as payment_count, SUM(amount) as total_payments
FROM Payment
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	payment_state text
1	Initial state - Payment counts:

```sql
RETURNING 'INSERTED Payment: ' || paymentid || ' for Loan ' || loanid

-- OPERATION 2: INSERT - Add another payment (Row 2)
INSERT INTO Payment (loanid, amount, paymentdate, mode)
VALUES (2, 150.00, '2024-03-11', 'Cash')
RETURNING 'INSERTED Payment: ' || paymentid || ' for Loan ' || loanid

-- OPERATION 3: UPDATE - Modify an existing payment (Row 3)
UPDATE Payment
SET amount = amount + 50.00
```

Data Output | Messages | Notifications

```
ROLLBACK

Query returned successfully in 74 msec.
```

Q4: Verify Denormalized Totals and Audit Logs

```sql

-- VERIFY DENORMALIZED TOTALS
SELECT '=== VERIFYING DENORMALIZED TOTALS ===' as verification;

-- Method 1: Compare denormalized vs actual totals
SELECT
```

Data Output | Messages | Notifications

Showing rows: 1 to 1 | Page No: 1 of 1

	verification text
1	=== VERIFYING DENORMALIZED TOTALS ===

```sql

-- Method 2: Show payment details for manual verification
SELECT 'Payment details for manual verification:' as payment_details;
SELECT
    p.loanid,
    p.paymentid,
    p.amount,
    p.paymentdate,
    p.mode
FROM Payment p
```

Data Output | Messages | Notifications

Showing rows: 1 to 1 | Page No: 1 of 1

	payment_details text
1	Payment details for manual verification:

```sql
-- Method 3: Loan summary with denormalized data
SELECT 'Loan summary with denormalized totals:' as loan_summary;
SELECT
    loanid,
    amount as original_loan_amount,
    total_payments as total_payments_received,
    amount - total_payments as remaining_balance,
    status
FROM Loan
ORDER BY loanid;
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	loan_summary text
1	Loan summary with denormalized totals:

```sql


-- SHOW AUDIT TABLE ENTRIES (2-3 ROWS EXPECTED)
SELECT '=== AUDIT TABLE ENTRIES (2-3 ROWS EXPECTED) ===' as audit_entr

SELECT
    audit_id,
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	audit_entries text
1	=== AUDIT TABLE ENTRIES (2-3 ROWS EXPECTED) ===

```sql
    audit_id,
    bef_total as before_total,
    aft_total as after_total,
    ROUND(aft_total - bef_total, 2) as net_change,
    key_col as loan_key,
    operation_type,
    rows_affected,
    changed_at
FROM Loan_AUDIT
ORDER BY changed_at;
```

Data Output Messages Notifications

	audit_id [PK] integer	before_total numeric (15,2)	after_total numeric (15,2)	net_change numeric	loan_key character varying (64)	operation_type character varying (10)	rows_affected integer	changed_at timestamp w

```sql
-- Audit entry analysis
SELECT
    'Audit Analysis:' as analysis,
    COUNT(*) as total_audit_entries,
    COUNT(DISTINCT key_col) as unique_loans_audited,
    SUM(rows_affected) as total_operations_logged,
    MIN(changed_at) as first_audit,
    MAX(changed_at) as last_audit
FROM Loan_AUDIT;
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	analysis text	total_audit_entries bigint	unique_loans_audited bigint	total_operations_logged bigint	first_audit timestamp without time zone	last_audit timestamp without time zon
1	Audit Analysis:	0	0	[null]	[null]	[null]

-Trigger Performance and Behavior Test

```sql
-- ADDITIONAL TEST: VERIFY TRIGGER BEHAVIOR
SELECT '=== ADDITIONAL TRIGGER BEHAVIOR TEST ===' as additional_test;
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	additional_test text
1	=== ADDITIONAL TRIGGER BEHAVIOR TEST ===

```sql
-- Test single operation to verify trigger fires correctly
BEGIN;
    INSERT INTO Payment (loanid, amount, paymentdate, mode)
    VALUES (1, 100.00, '2024-03-12', 'Online')
    RETURNING 'Test Insert - Payment ID: ' || paymentid as test_result;

    -- Check audit entry was created
    SELECT 'Audit entries after test insert:' as audit_check;
    SELECT audit_id, bef_total, aft_total, key_col, operation_type
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	audit_check text
1	Audit entries after test insert:

```sql
    SELECT audit_id, bef_total, aft_total, key_col, operation_type
    FROM Loan_AUDIT
    ORDER BY changed_at DESC
    LIMIT 1;
ROLLBACK; -- Rollback to avoid affecting row count

-- Verify rollback worked (no new permanent rows)
SELECT 'After rollback - Payment count:', COUNT(*) FROM Payment;
SELECT 'After rollback - Audit entries:', COUNT(*) FROM Loan_AUDIT;
```

Data Output Messages Notifications

	audit_id [PK] integer	bef_total numeric (15,2)	aft_total numeric (15,2)	key_col character varying (64)	operation_type character varying (10)


```sql
    SELECT audit_id, bef_total, aft_total, key_col, operation_type
    FROM Loan_AUDIT
    ORDER BY changed_at DESC
    LIMIT 1;
ROLLBACK; -- Rollback to avoid affecting row count

-- Verify rollback worked (no new permanent rows)
SELECT 'After rollback - Payment count:', COUNT(*) FROM Payment;
SELECT 'After rollback - Audit entries:', COUNT(*) FROM Loan_AUDIT;
```


Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	?column? text	count bigint
1	After rollback - Audit entries:	0

B8: Recursive Hierarchy Roll-Up

Q1: Create HIER Table with Banking Domain Context

```sql
-- CREATE HIERARCHY TABLE FOR BANKING DOMAIN
CREATE TABLE HIER (
    parent_id VARCHAR(50),
    child_id VARCHAR(50) PRIMARY KEY,
    relationship_type VARCHAR(50),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
select * from HIER;
```

Data Output Messages Notifications

parent_id character varying (50)	child_id [PK] character varying (50)	relationship_type character varying (50)	created_at timestamp without time zone

```sql
-- Verify table creation
SELECT 'HIER table created successfully' as status;
SELECT column_name, data_type, is_nullable
FROM information_schema.columns
WHERE table_name = 'hier'
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of

	status text
1	HIER table created successfully

```sql
-- Verify table creation
SELECT 'HIER table created successfully' as status;
SELECT column_name, data_type, is_nullable
FROM information_schema.columns
WHERE table_name = 'hier'
ORDER BY ordinal_position;
```

Data Output Messages Notifications

Showing rows: 1 to 4 Page No: 1 of 1

	column_name name	data_type character varying	is_nullable character varying (3)
1	parent_id	character varying	YES
2	child_id	character varying	NO
3	relationship_type	character varying	YES
4	created_at	timestamp without time zone	YES

Q2: Insert 6-10 Rows Forming 3-Level Banking Hierarchy

```sql
INSERT INTO HIER (parent_id, child_id, relationship_type) VALUES
-- Level 1: Root accounts (Corporate Structure)
(NULL, 'BANK_ROOT', 'Financial Institution'),

-- Level 2: Main banking divisions
('BANK_ROOT', 'RETAIL_BANKING', 'Division'),
('BANK_ROOT', 'BUSINESS_BANKING', 'Division'),
('BANK_ROOT', 'INVESTMENT_BANKING', 'Division'),

-- Level 3: Sub-divisions under Retail Banking
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	insertion_status text
1	Hierarchy data inserted: 0 rows

```sql
-- Level 3: Sub-divisions under Business Banking
('BUSINESS_BANKING', 'CORPORATE_ACCOUNTS', 'Department'),
('BUSINESS_BANKING', 'COMMERCIAL_LOANS', 'Department');

-- Verify insertion
SELECT 'Hierarchy data inserted: ' || COUNT(*) || ' rows' as insertion_

-- Show the hierarchy structure
SELECT 'Hierarchy Structure:' as structure;
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	structure text
1	Hierarchy Structure:

```sql
    relationship_type,
    created_at
FROM HIER
ORDER BY
    CASE
        WHEN parent_id IS NULL THEN 0
        WHEN parent_id = 'BANK_ROOT' THEN 1
        ELSE 2
    END,
    parent_id,
```

Data Output Messages Notifications

parent_id character varying (50)	child_id [PK] character varying (50)	relationship_type character varying (50)	created_at timestamp without time zone

Q3: Create Recursive WITH Query for Hierarchy Roll-Up

```
2737
2738  URSIVE HIERARCHY QUERY WITH TRANSACTION ROLL-UP
2739   '=== RECURSIVE HIERARCHY WITH TRANSACTION ROLL-UP ===' as query_title;
2740
2741  ECURSIVE account_hierarchy AS (
2742   Anchor: Root level
2743  LECT
2744    child_id,
2745    child_id as root_id,
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	query_title text
1	=== RECURSIVE HIERARCHY WITH TRANSACTION ROLL-UP ===

```
2737
2738  -- RECURSIVE HIERARCHY QUERY WITH TRANSACTION ROLL-UP
2739  SELECT '=== RECURSIVE HIERARCHY WITH TRANSACTION ROLL-UP ===' as query_
2740
2741  -- PRACTICAL BANKING HIERARCHY ROLL-UP WITH ACTUAL TRANSACTIONS
2742  SELECT '=== PRACTICAL BANKING HIERARCHY ROLL-UP ===' as query_title;
2743
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No:

	query_title text
1	=== PRACTICAL BANKING HIERARCHY ROLL-UP ===

```
2743
2744  -- SIMPLE RECURSIVE QUERY (6-10 ROWS OUTPUT)
2745  SELECT '=== SIMPLE RECURSIVE HIERARCHY TRAVERSAL ===' as query_title;
2746  -- FINAL WORKING RECURSIVE HIERARCHY QUERY (6-10 ROWS)
2747  SELECT '=== FINAL RECURSIVE HIERARCHY IMPLEMENTATION ===' as implementa
2748
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	query_title text
1	=== SIMPLE RECURSIVE HIERARCHY TRAVERSAL ===

```sql
-- SIMPLE RECURSIVE QUERY (6-10 ROWS OUTPUT)
SELECT '=== SIMPLE RECURSIVE HIERARCHY TRAVERSAL ===' as query_title;
-- FINAL WORKING RECURSIVE HIERARCHY QUERY (6-10 ROWS)
SELECT '=== FINAL RECURSIVE HIERARCHY IMPLEMENTATION ===' as implementa
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	implementation text
1	=== FINAL RECURSIVE HIERARCHY IMPLEMENTATION ===

Q4 .Reuse existing seed rows; do not exceed the ≤10 committed rows budget.

```sql
-- FINAL BUDGET VERIFICATION (≤10 ROWS)
SELECT '=== FINAL ROW BUDGET VERIFICATION ===' as budget_check;

WITH all_tables AS (
    SELECT 'hier' as table_name, COUNT(*) as row_count FROM hier
    UNION ALL SELECT 'loan', COUNT(*) FROM loan
    UNION ALL SELECT 'payment', COUNT(*) FROM payment
    UNION ALL SELECT 'loan_audit', COUNT(*) FROM loan_audit
    UNION ALL SELECT 'local_transaction_audit', COUNT(*) FROM local_tra
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	budget_check text
1	=== FINAL ROW BUDGET VERIFICATION ===

B9: Mini-Knowledge Base with Transitive Inference

Q1: Create TRIPLE Table and Insert Facts

```sql
-- CREATE TRIPLE TABLE FOR KNOWLEDGE BASE
CREATE TABLE TRIPLE (
    s VARCHAR(64),
    p VARCHAR(64),
    o VARCHAR(64),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (s, p, o)
```

Data Output | Messages | Notifications

```
CREATE TABLE

Query returned successfully in 88 msec.
```

```sql
; rule implications
:Account', 'isA', 'SavingsAccount'),
:Account', 'requires', 'EnhancedDueDiligence'),
-equires', 'CreditCheck'),
 'canHave', 'Overdraft');

nsertion
:PLE table created with ' || COUNT(*) || ' facts' as status FROM TRIPLE;

 facts
```

Data Output | Messages | Notifications

Showing rows: 1 to 1 | Page No: 1 | of 1

	status text
1	TRIPLE table created with 0 facts

```sql
('HighValueAccount', 'requires', 'EnhancedDueDiligence'),
('Loan', 'requires', 'CreditCheck'),
('Account', 'canHave', 'Overdraft');

-- Verify insertion
SELECT 'TRIPLE table created with ' || COUNT(*) || ' facts' as status F

-- Show all facts
SELECT 'Knowledge Base Facts (10 facts):' as facts;
SELECT s, p, o FROM TRIPLE ORDER BY p, s;
```

Data Output | Messages | Notifications

Showing rows: 1 to 1 | Page No: 1 | of 1

	facts text
1	Knowledge Base Facts (10 facts):

```
('HighValueAccount', 'requires', 'EnhancedDueDiligence'),
('Loan', 'requires', 'CreditCheck'),
('Account', 'canHave', 'Overdraft');

-- Verify insertion
SELECT 'TRIPLE table created with ' || COUNT(*) || ' facts' as status

-- Show all facts
SELECT 'Knowledge Base Facts (10 facts):' as facts;
SELECT s, p, o FROM TRIPLE ORDER BY p, s;
```

Data Output Messages Notifications

s [PK] character varying (64)	p [PK] character varying (64)	o [PK] character varying (64)

Q2.Insert 8–10 domain facts relevant to your project (e.g., simple type hierarchy or rule implications).

```
---------
-- INSERT 10 BANKING DOMAIN FACTS INTO TRIPLE TABLE
SELECT '=== INSERTING BANKING DOMAIN FACTS INTO TRIPLE TABLE ===' as st

-- Clear any existing data to ensure clean state
DELETE FROM TRIPLE;
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	step text
1	=== INSERTING BANKING DOMAIN FACTS INTO TRIPLE TABLE ===

```
-- Insert 10 banking domain facts covering type hierarchy and business
INSERT INTO TRIPLE (s, p, o) VALUES
-- TYPE HIERARCHY FACTS (6 facts)
('SavingsAccount', 'isA', 'Account'),
('CurrentAccount', 'isA', 'Account'),
('FixedDepositAccount', 'isA', 'Account'),
('PersonalLoan', 'isA', 'Loan'),
('BusinessLoan', 'isA', 'Loan'),
('Account', 'isA', 'BankProduct'),
('Loan', 'isA', 'BankProduct'),
```

Data Output Messages Notifications

```
DELETE 0

Query returned successfully in 92 msec.
```

```
2938
2939  e have exactly 10 facts
2940  erted ' || COUNT(*) || ' banking domain facts' as verification FROM TRIF
2941
2942  all facts organized by predicate
2943   COMPLETE KNOWLEDGE BASE FACTS ===' as display;
2944
2945  lationship_type,
2946  bject,
2947  ject,
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	verification text
1	Inserted 0 banking domain facts

```
2936  -- RISK CLASSIFICATION (1 fact)
2937  ('SuspiciousActivity', 'classifiedAs', 'HighRisk');
2938
2939  -- Verify we have exactly 10 facts
2940  SELECT 'Inserted ' || COUNT(*) || ' banking domain facts' as verificatic
2941
2942  -- Display all facts organized by predicate
2943  SELECT '=== COMPLETE KNOWLEDGE BASE FACTS ===' as display;
2944  SELECT
2945      p as relationship_type,
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	display text
1	=== COMPLETE KNOWLEDGE BASE FACTS ===

```
2956  ORDER BY
2957      CASE p
2958          WHEN 'isA' THEN 1
2959          WHEN 'requires' THEN 2
2960          WHEN 'triggers' THEN 3
2961          WHEN 'classifiedAs' THEN 4
2962          ELSE 5
2963      END,
2964      s,
2965      o;
```

Data Output Messages Notifications

	relationship_type character varying (64)	subject character varying (64)	object character varying (64)	category text

```sql
------
-- CLEANUP TEMPORARY DATA TO MAINTAIN ≤10 ROW BUDGET
SELECT '=== CLEANING UP TEMPORARY DATA ===' as cleanup;

-- Option 1: Remove audit tables (most likely to have temporary data)
DROP TABLE IF EXISTS loan_audit CASCADE;
DROP TABLE IF EXISTS local_transaction_audit CASCADE;
```

Data Output | Messages | Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	cleanup text
1	=== CLEANING UP TEMPORARY DATA ===

```sql
DROP TABLE IF EXISTS local_transaction_audit CASCADE;

-- Option 2: Clean specific test data from core tables
DELETE FROM payment WHERE mode LIKE '%Test%' OR mode LIKE '%Simulation%
DELETE FROM loan WHERE status LIKE '%Test%' OR description LIKE '%Simu

-- Option 3: Reset Transaction tables to minimal dataset
DELETE FROM Transaction_A WHERE TransID > 5;
DELETE FROM transaction_b WHERE TransID > 5;
```

Data Output | Messages | Notifications

```
DELETE 5

Query returned successfully in 87 msec.
```

```sql
-- Option 3: Reset Transaction tables to minimal dataset
DELETE FROM Transaction_A WHERE TransID > 5;
DELETE FROM transaction_b WHERE TransID > 5;

-- Verify cleanup
```

Data Output | Messages | Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	cleanup_status text
1	After cleanup - Removed temporary audit and test data

```sql
-- FINAL OPTIMIZED SETUP WITHIN 10-ROW BUDGET
SELECT '=== FINAL OPTIMIZED SETUP (≤10 ROWS) ===' as final_setup;

-- Step 1: Ensure TRIPLE has exactly 8 facts (as required)
DELETE FROM TRIPLE;
INSERT INTO TRIPLE (s, p, o) VALUES
```

Data Output | Messages | Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	final_setup text
1	=== FINAL OPTIMIZED SETUP (≤10 ROWS) ===

```sql
INSERT INTO HIER (parent_id, child_id, relationship_type) VALUES
(NULL, 'BANK_ROOT', 'Financial Institution'),
('BANK_ROOT', 'RETAIL_BANKING', 'Division'),
('BANK_ROOT', 'BUSINESS_BANKING', 'Division'),
('BANK_ROOT', 'INVESTMENT_BANKING', 'Division'),
('RETAIL_BANKING', 'PERSONAL_ACCOUNTS', 'Department'),
('RETAIL_BANKING', 'LOAN_SERVICES', 'Department'),
('BUSINESS_BANKING', 'CORPORATE_ACCOUNTS', 'Department'),
('BUSINESS_BANKING', 'COMMERCIAL_LOANS', 'Department');
```

Data Output | Messages | Notifications

```
INSERT 0 8

Query returned successfully in 72 msec.
```

```sql
(2, 1004, 'T002', 250.75, 'Debit', '2024-01-16');

DELETE FROM transaction_b;
INSERT INTO transaction_b (TransID, AccountID, TellerID, Amount, Type,
(1, 1001, 'T002', 2000.00, 'Credit', '2024-01-10'),
(2, 1003, 'T003', 150.00, 'Debit', '2024-01-11'),
(3, 1005, 'T001', 4500.75, 'Credit', '2024-01-12');

-- Verify final row counts
SELECT 'Final row counts after optimization:' as verification;
```

Data Output | Messages | Notifications

Showing rows: 1 to 1 | Page No: 1 of 1

	verification text
1	Final row counts after optimization:

B10: Business Limit Alert (Function + Trigger)

Step 1: Create BUSINESS_LIMITS Table and Seed Active Rule

```sql
-- CREATE BUSINESS_LIMITS TABLE
CREATE TABLE IF NOT EXISTS BUSINESS_LIMITS (
    rule_key VARCHAR(64) PRIMARY KEY,
    threshold DECIMAL(15,2),
    active CHAR(1) CHECK (active IN ('Y', 'N')),
    description TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Data Output | Messages | Notifications

```
CREATE TABLE

Query returned successfully in 83 msec.
```

```sql
-- SEED EXACTLY ONE ACTIVE RULE (Payment Amount Limit)
INSERT INTO BUSINESS_LIMITS (rule_key, threshold, active, description)
VALUES ('MAX_PAYMENT_AMOUNT', 2000.00, 'Y', 'Maximum allowed amount fo
ON CONFLICT (rule_key) DO UPDATE SET
    threshold = EXCLUDED.threshold,
    active = EXCLUDED.active,
    description = EXCLUDED.description;
```

Data Output | Messages | Notifications

```
INSERT 0 1

Query returned successfully in 172 msec.
```

```
 active rule
ESS_LIMITS table created with active rule:' as verification;
ey, threshold, active, description FROM BUSINESS_LIMITS;
```

Data Output | Messages | Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	verification text
1	BUSINESS_LIMITS table created with active rule:

```
USINESS_LIMITS (rule_key, threshold, active, description)
PAYMENT_AMOUNT', 2000.00, 'Y', 'Maximum allowed amount for a single pay
rule_key) DO UPDATE SET
 = EXCLUDED.threshold,
EXCLUDED.active,
on = EXCLUDED.description;

 active rule
ESS_LIMITS table created with active rule:' as verification;
ey, threshold, active, description FROM BUSINESS_LIMITS;
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	rule_key [PK] character varying (64)	threshold numeric (15,2)	active character (1)	description text
1	MAX_PAYMENT_AMOUNT	2000.00	Y	Maximum allowed amount for a single payment

Q2

```

    RETURN 0; -- No violation
END;
$$ LANGUAGE plpgsql;

-- Test the function
SELECT 'Testing alert function with various amounts:' as function_test;
SELECT
    fn_should_alert_payment(1500.00, 1) as test_1500,  -- Should pass (
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	function_test text
1	Testing alert function with various amounts:

No limit

Query Query History Scratch Pad

```
BEFORE INSERT OR UPDATE ON payment
FOR EACH ROW
EXECUTE FUNCTION check_payment_limits();

-- Verify trigger creation
SELECT 'Business limit trigger created on payment table:' as trigger_st
SELECT
    trigger_name,
    event_manipulation,
    action_timing,
    action_statement
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	trigger_status text
1	Business limit trigger created on payment table:

```
---------
-- DEMONSTRATE 2 FAILING AND 2 PASSING DML CASES
SELECT '=== DEMONSTRATING BUSINESS LIMIT ENFORCEMENT ===' as demonstrat

-- Get current state for verification
SELECT 'Current payment count before tests:', COUNT(*) FROM payment;
```

Data Output | Messages | Notifications

Showing rows: 1 to 1 | Page No: 1 of 1

	demonstration text
1	=== DEMONSTRATING BUSINESS LIMIT ENFORCEMENT ===

Query | Query History | Scratch Pad

```
EGIN;
   -- TEST 1: PASSING INSERT - Within limit
   SELECT 'Test 1: Payment within limit (Should PASS)' as test_case;
   INSERT INTO payment (loanid, amount, paymentdate, mode)
   VALUES (1, 500.00, CURRENT_DATE, 'Cash')
   RETURNING '✅ PASS: Payment inserted - Amount: ' || amount as result

   -- TEST 2: PASSING INSERT - At the limit boundary
   SELECT 'Test 2: Payment at limit boundary (Should PASS)' as test_case
   INSERT INTO payment (loanid, amount, paymentdate, mode)
```

Data Output | Messages | Notifications

```
BEGIN

Query returned successfully in 84 msec.
```

```
   -- Show successful transactions that will be committed
   SELECT 'Successful payments to be committed:' as committed_data;
   SELECT paymentid, loanid, amount, paymentdate, mode
   FROM payment
   ORDER BY paymentid DESC
   LIMIT 5;
```

Data Output | Messages | Notifications

Showing rows: 1 to 1 | Page No: 1 of 1

	committed_data text
1	Successful payments to be committed:

```
-----
- CLEANUP TO ENSURE ≤10 ROW BUDGET
' '=== FINAL CLEANUP FOR BUDGET MAINTENANCE ===' as cleanup;
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	cleanup text
1	=== FINAL CLEANUP FOR BUDGET MAINTENANCE ===

```
-- Verify final state
SELECT 'Final payment count after cleanup:', COUNT(*) FROM payment;
SELECT 'Final business limits count:', COUNT(*) FROM business_limits;

-- Final budget confirmation
SELECT
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	?column? text	count bigint
1	Final business limits count:	1