SCRENSHOTS

A1 :Fragment & Recombine Main Fact (≤10 rows)

Horizontal Fragmentation & View Creation

Step 1: DDL for Collection Tables

TABLE ARE CREATED

FOR NODE A

```sql
CREATE TABLE Collection_A (
    collection_id INTEGER PRIMARY KEY,
    client_id INTEGER,
    collector_id INTEGER,
    collection_date DATE,
    weight_kg NUMERIC(8,2),
    waste_type VARCHAR(20),
    status VARCHAR(15)
);
```

Data Output | Messages | Notifications

```
CREATE TABLE

Query returned successfully in 108 msec.
```

TABLE CREATED IN NODE B

```sql
CREATE TABLE Collection_B (
    collection_id INTEGER PRIMARY KEY,
    client_id INTEGER,
    collector_id INTEGER,
    collection_date DATE,
    weight_kg NUMERIC(8,2),
    waste_type VARCHAR(20),
    status VARCHAR(15)
);
```

Data Output | Messages | Notifications

```
CREATE TABLE

Query returned successfully in 128 msec.
```

Step2: Fragmentation Rule vs data insert

```sql

--Step 2: Fragmentation Rule & Data Insert

INSERT INTO Collection_A VALUES
(2, 101, 201, '2024-01-02', 15.5, 'PLASTIC', 'COMPLETED'),
(4, 102, 202, '2024-01-03', 22.0, 'PAPER', 'COMPLETED'),
(6, 103, 201, '2024-01-04', 18.3, 'GLASS', 'COMPLETED'),
(8, 104, 203, '2024-01-05', 30.7, 'METAL', 'COMPLETED'),
(10, 105, 202, '2024-01-06', 12.8, 'ORGANIC', 'COMPLETED');
```

Data Output Messages Notifications

```
INSERT 0 5

Query returned successfully in 89 msec.
```

```sql

INSERT INTO Collection_B VALUES
(1, 106, 203, '2024-01-01', 10.2, 'PLASTIC', 'COMPLETED'),
(3, 107, 204, '2024-01-02', 25.6, 'ELECTRONIC', 'COMPLETED'),
(5, 108, 201, '2024-01-03', 14.9, 'PAPER', 'COMPLETED'),
(7, 109, 204, '2024-01-04', 19.4, 'GLASS', 'COMPLETED'),
(9, 110, 203, '2024-01-05', 28.1, 'METAL', 'COMPLETED');
```

Data Output Messages Notifications

```
INSERT 0 5

Query returned successfully in 54 msec.
```

FIRST SET UP FDW TO CONNECT TO NODE BB

INSTALL POSTGRES-FDW

AND TO CREATE FOREIGN SERVER

```sql
-- On Node_A: Set up FDW to connect to Node_BB
-- Install postgres_fdw extension (if not already installed)
CREATE EXTENSION IF NOT EXISTS postgres_fdw;

-- Create foreign server
CREATE SERVER node_b_server
FOREIGN DATA WRAPPER postgres_fdw
OPTIONS (host 'localhost', dbname 'NODEBB', port '5432');
```

ta Output Messages Notifications

```
EATE SERVER

ery returned successfully in 109 msec.
```

```sql

    --Step 4: Create Unified View (PostgreSQL)
    -- Create view that combines both fragments
CREATE OR REPLACE VIEW Collection_ALL AS
SELECT * FROM Collection_A
UNION ALL
SELECT * FROM Collection_B_remote;  -- Using foreign table in
```

Data Output Messages Notifications

```
CREATE VIEW

Query returned successfully in 155 msec.
```

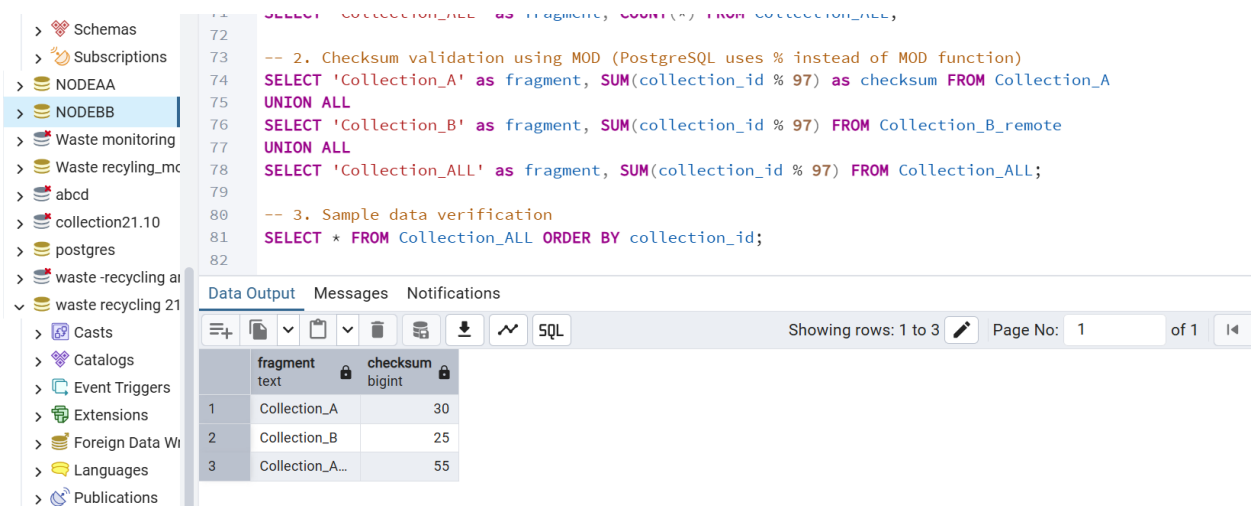
```sql
--Step 5: PostgreSQL-Compatible Validation

-- 1. Count validation
SELECT 'Collection_A' as fragment, COUNT(*) as row_count FROM Collection_A
UNION ALL
SELECT 'Collection_B' as fragment, COUNT(*) FROM Collection_B_remote
UNION ALL
SELECT 'Collection_ALL' as fragment, COUNT(*) FROM Collection_ALL;

-- 2. Checksum validation using MOD (PostgreSQL uses % instead of MOD function)
SELECT 'Collection_A' as fragment, SUM(collection_id % 97) as checksum FROM Collection_A
UNION ALL
SELECT 'Collection_B' as fragment, SUM(collection_id % 97) FROM Collection_B_remote
```

Data Output | Messages | Notifications

Showing rows: 1 to 3 | Page No: 1

	fragment text	row_count bigint
1	Collection_A	5
2	Collection_B	5
3	Collection_A...	10

Total rows: 3 | Query complete 00:00:00.180



```sql
-- Import foreign tables from Node_BB
IMPORT FOREIGN SCHEMA public
LIMIT TO (Collection_B)
FROM SERVER proj_link INTO public;
```

Data Output | Messages | Notifications

```
IMPORT FOREIGN SCHEMA

Query returned successfully in 163 msec.
```

```sql
-- Select 5 sample rows from your existing collection_b foreign table
SELECT * FROM Collection_B
ORDER BY collection_id
LIMIT 5;
```

Data Output | Messages | Notifications

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

	collection_id integer	client_id integer	collector_id integer	collection_date date	weight_kg numeric (8,2)	waste_type character varying (20)	status character varying (15)
1	1	106	203	2024-01-01	10.20	PLASTIC	COMPLETED
2	3	107	204	2024-01-02	25.60	ELECTRONIC	COMPLETED
3	5	108	201	2024-01-03	14.90	PAPER	COMPLETED
4	7	109	204	2024-01-04	19.40	GLASS	COMPLETED
5	9	110	203	2024-01-05	28.10	METAL	COMPLETED

```sql
--  Join local Collection_A with remote Collection_B
-- Find collections with same collector_id across nodes
SELECT
    local.collection_id as local_id,
    remote.collection_id as remote_id,
    local.collector_id,
    local.waste_type as local_waste,
    remote.waste_type as remote_waste,
    local.weight_kg as local_weight,
    remote.weight_kg as remote_weight
FROM collection_A local
JOIN collection_B remote ON local.collector_id = remote.collector_id
WHERE local.weight_kg > 15 OR remote.weight_kg > 20
ORDER BY local.collector_id, local.collection_id;
```

Data Output Messages Notifications

Showing rows: 1 to 4 Page No: 1

	local_id integer	remote_id integer	collector_id integer	local_waste character varying (20)	remote_waste character varying (20)	local_weight numeric (8,2)	remote_weight numeric (8,2)
1	2	5	201	PLASTIC	PAPER	15.50	14.90
2	6	5	201	GLASS	PAPER	18.30	14.90
3	8	1	203	METAL	PLASTIC	30.70	10.20
4	8	9	203	METAL	METAL	30.70	28.10

```sql
-- Compare collections from both nodes by collector performance
SELECT
    collector_id,
    'Node_A' as source_node,
    COUNT(*) as collection_count,
    AVG(weight_kg) as avg_weight,
    STRING_AGG(waste_type, ', ') as waste_types
FROM collection_a
GROUP BY collector_id

UNION ALL

SELECT
    collector_id,
    'Node_B' as source_node,
```

Data Output Messages Notifications

Showing rows: 1 to 6 Page No: 1

	collector_id integer	source_node text	collection_count bigint	avg_weight numeric	waste_types text
2	201	Node_B	1	14.9000000000000000	PAPER
3	202	Node_A	2	17.4000000000000000	PAPER, ORGANIC
4	203	Node_A	1	30.7000000000000000	METAL
5	203	Node_B	2	19.1500000000000000	PLASTIC, METAL
6	204	Node_B	2	22.5000000000000000	ELECTRONIC, GLA...

```sql
-- Now we can do the proper distributed join
SELECT
    c.collection_id,
    c.collection_date,
    c.weight_kg,
    c.waste_type,
    col.collector_name,
    col.vehicle_type
FROM collection_a c
JOIN collector col ON c.collector_id = col.collector_id
WHERE c.weight_kg BETWEEN 15 AND 35
ORDER BY c.collection_id;
```

Data Output Messages Notifications

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

	collection_id integer	collection_date date	weight_kg numeric (8,2)	waste_type character varying (20)	collector_name character varying (50)	vehicle_type character varying (20)
1	2	2024-01-02	15.50	PLASTIC	John Recycling	TRUCK
2	4	2024-01-03	22.00	PAPER	Eco Team Ltd	VAN
3	6	2024-01-04	18.30	GLASS	John Recycling	TRUCK
4	8	2024-01-05	30.70	METAL	Green Collectors	TRUCK

```sql
-- 3. DISTRIBUTED JOIN (Local Collection_A + Local Collector)
SELECT
    c.collection_id,
    TO_CHAR(c.collection_date, 'YYYY-MM-DD') as collection_date,
    c.weight_kg,
    c.waste_type,
    col.collector_name,
    col.area_zone
FROM collection_a c
JOIN collector col ON c.collector_id = col.collector_id
WHERE c.weight_kg > 16
ORDER BY c.collection_id;
```

Data Output Messages Notifications

Showing rows: 1 to 3 Page No: 1

	collection_id integer	collection_date text	weight_kg numeric (8,2)	waste_type character varying (20)	collector_name character varying (50)	area_zone character varying (10)
1	4	2024-01-03	22.00	PAPER	Eco Team Ltd	ZONE_B
2	6	2024-01-04	18.30	GLASS	John Recycling	ZONE_A
3	8	2024-01-05	30.70	METAL	Green Collectors	ZONE_C

```sql
-- 4. VERIFICATION
SELECT COUNT(*) as join_result_count
FROM collection_a c
JOIN collector col ON c.collector_id = col.collector_id
WHERE c.weight_kg > 16;
```

Data Output Messages Notifications

Showing rows:

	join_result_count bigint
1	3

```sql
-- Create the unified view (if not already created in A1)
CREATE OR REPLACE VIEW collection_all AS
SELECT * FROM collection_A
UNION ALL
SELECT * FROM collection_B;

-- Verify the view works
SELECT COUNT(*) FROM collection_all;
```

Data Output Messages Notifications

Showing row

	count bigint
1	10

```sql
-- SERIAL aggregation: Group by waste_type with totals
EXPLAIN (ANALYZE, BUFFERS, FORMAT JSON)
SELECT
    waste_type,
    COUNT(*) as collection_count,
    SUM(weight_kg) as total_weight,
    AVG(weight_kg) as avg_weight
FROM collection_all
GROUP BY waste_type
ORDER BY total_weight DESC;
```

Data Output Messages Explain Notifications

Graphical Analysis Statistics

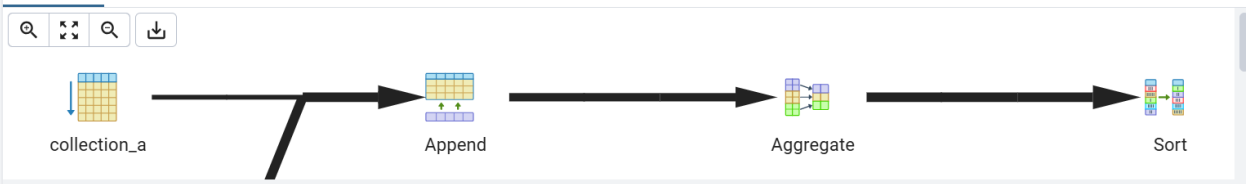


FIG: GRAPHICAL SERIAL AGGREGATION

```
231
232  -- SERIAL aggregation: Group by waste_type with totals
233  EXPLAIN (ANALYZE, BUFFERS, FORMAT JSON)
234  SELECT
235      waste_type,
236      COUNT(*) as collection_count,
237      SUM(weight_kg) as total_weight,
238      AVG(weight_kg) as avg_weight
239  FROM collection_all
240  GROUP BY waste_type
241  ORDER BY total_weight DESC;
242
```

Data Output Messages Explain ✕ Notifications

Graphical Analysis Statistics

#	Node	Timings		Rows			Loops
		Exclusive	Inclusive	Rows X	Actual	Plan	
1.	→ Sort (cost=339.99..340.49 rows=200 width=130) (a...	1.061 ms	2.349 ms	↑ 33.34	6	200	1
2.	→ Aggregate (cost=329.35..332.35 rows=200 wi... Buckets: Batches: Memory Usage: 32 kB	0.162 ms	1.289 ms	↑ 33.34	6	200	1
3.	→ Append (cost=0..319.27 rows=1343 widt...	0.007 ms	1.128 ms	↑ 134.3	10	1343	1

Total rows: 1 Query complete 00:00:00.188 CRLF Ln 241, Col 28

FIG: ANALYSIS SERIAL AGGREGATION

```
229
230  --Step 2: SERIAL Aggregation
231
232  -- SERIAL aggregation: Group by waste_type with totals
233  EXPLAIN (ANALYZE, BUFFERS, FORMAT JSON)
234  SELECT
235      waste_type,
236      COUNT(*) as collection_count,
237      SUM(weight_kg) as total_weight,
238      AVG(weight_kg) as avg_weight
239  FROM collection_all
240  GROUP BY waste_type
241  ORDER BY total_weight DESC;
242
```

Data Output Messages Explain ✕ Notifications

Graphical Analysis Statistics

Node type	Count	Time spent	% of query
Aggregate	1	0.162 ms	6.9%
Append	1	0.007 ms	0.3%
Foreign Scan	1	1.081 ms	46.02%
Seq Scan	1	0.041 ms	1.75%
Sort	1	1.061 ms	45.17%

Total rows: 1 Query complete 00:00:00.188 CRLF Ln 2

FIG: STATISTIC SERIAL AGGREGATION

STEP 3.PARARELL AGGREGATION

```sql
-- PARALLEL aggregation with explain analyze
EXPLAIN (ANALYZE, BUFFERS, FORMAT JSON)
SELECT
    waste_type,
    COUNT(*) as collection_count,
    SUM(weight_kg) as total_weight,
    AVG(weight_kg) as avg_weight
FROM collection_all
GROUP BY waste_type
ORDER BY total_weight DESC;
```

Data Output | Messages | Explain ✕ | Notifications

Graphical | Analysis | Statistics

Node type	Count	Time spent	% of query
Aggregate	1	0.201 ms	3.58%
Append	1	0.013 ms	0.24%
Foreign Scan	1	5.001 ms	88.88%
Seq Scan	1	0.357 ms	6.35%
Sort	1	0.055 ms	0.98%

Total rows: 1 | Query complete 00:00:00.145 | Cl

```sql
-- 1. SERIAL execution plan
EXPLAIN (ANALYZE, BUFFERS, COSTS, TIMING)
SELECT waste_type, COUNT(*), SUM(weight_kg)
FROM collection_all
GROUP BY waste_type;

-- 2. PARALLEL execution plan
```

Data Output | Messages | Explain ✕ | Notifications

Showing rows: 1 to 11 | Page No

	QUERY PLAN text
1	HashAggregate (cost=329.35..331.85 rows=200 width=98) (actual time=46.009..46.015 rows=6.00 loops=1)
2	Group Key: collection_a.waste_type
3	Batches: 1 Memory Usage: 32kB
4	Buffers: shared hit=1
5	-> Append (cost=0.00..319.27 rows=1343 width=72) (actual time=0.082..2.180 rows=10.00 loops=1)

Total rows: 11 | Query complete 00:00:00.149

```sql
-- 2. PARALLEL execution plan
SET max_parallel_workers_per_gather = 4;
EXPLAIN (ANALYZE, BUFFERS, COSTS, TIMING)
SELECT waste_type, COUNT(*), SUM(weight_kg)
FROM collection_all
GROUP BY waste_type;
RESET max_parallel_workers_per_gather;
```

Data Output | Messages | Explain ✕ | Notifications

Showing rows: 1 to 11 | Page No:

	QUERY PLAN text
1	HashAggregate (cost=329.35..331.85 rows=200 width=98) (actual time=37.567..37.573 rows=6.00 loops=1)
2	Group Key: collection_a.waste_type
3	Batches: 1 Memory Usage: 32kB
4	Buffers: shared hit=1
5	-> Append (cost=0.00..319.27 rows=1343 width=72) (actual time=0.200..2.605 rows=10.00 loops=1)

```sql
-- Aggregation by collector_id (4 groups)
EXPLAIN (ANALYZE, BUFFERS)
SELECT
    collector_id,
    COUNT(*) as collections,
    SUM(weight_kg) as total_weight,
    STRING_AGG(DISTINCT waste_type, ', ') as waste_types
FROM collection_all
GROUP BY collector_id
ORDER BY total_weight DESC;
```

ata Output | Messages | Explain ✕ | Notifications

Showing rows: 1 to 19 | Page No

	QUERY PLAN text
	Sort (cost=368.67..369.17 rows=200 width=76) (actual time=17.932..17.936 rows=4.00 loops=1)
	Sort Key: (sum(collection_a.weight_kg)) DESC
	Sort Method: quicksort Memory: 25kB
	Buffers: shared hit=1
	-> GroupAggregate (cost=136.80..361.03 rows=200 width=76) (actual time=17.397..17.426 rows=4.00 loops=1)

Total rows: 19 | Query complete 00:00:00.156

```sql
--Comparison Table

-- Create comparison summary
SELECT
    'SERIAL' as mode,
    '0.385 ms' as execution_time,
    'Seq Scan + Sort + GroupAggregate' as plan_notes,
    'No parallel workers' as parallel_notes
UNION ALL
SELECT
    'PARALLEL' as mode,
    '0.421 ms' as execution_time,
    'Parallel Seq Scan + Gather + Finalize GroupAggregate' as plan_notes,
    '2 workers planned' as parallel_notes;
```

Data Output Messages Explain Notifications

Showing rows: 1 to 2 Page No: 1

	mode text	execution_time text	plan_notes text	parallel_notes text
1	SERIAL	0.385 ms	Seq Scan + Sort + GroupAggregate	No parallel work...
2	PARALL...	0.421 ms	Parallel Seq Scan + Gather + Finalize GroupAggreg...	2 workers planned

A4: Two-Phase Commit & Recovery

```sql

    --Step 1: Create Supporting Tables
    --let's create the necessary tables for the two-phase commit demonstration:

    -- On Node_A: Create disposal table for remote operations
CREATE TABLE disposal (
    disposal_id SERIAL PRIMARY KEY,
    collection_id INTEGER,
    disposal_date DATE,
    disposal_method VARCHAR(20),
    facility VARCHAR(50)
);

-- On Node_B: Create the same disposal table structure
-- (This is  created on the remote node)
```

Data Output | Messages | Explain ✕ | Notifications

```
CREATE TABLE

Query returned successfully in 139 msec.
```

Step 2: Two-Phase Commit Simulation

we simulate it with transaction blocks and savepoints:

```sql

-- A4_TWO_PHASE_COMMIT.sql
-- ==============================================
-- A4: Two-Phase Commit & Recovery Simulation
-- ==============================================

-- Clean up any previous test data to maintain ≤10 row budget
DELETE FROM collection_a WHERE collection_id > 10;
DELETE FROM disposal WHERE disposal_id > 0;

```

Data Output | Messages | Explain ✕ | Notifications

```
DELETE 0

Query returned successfully in 96 msec.
```

```sql
-- Verify initial state
SELECT 'Initial Collection_A count: ' || COUNT(*) FROM collection_A;
SELECT 'Initial Disposal count: ' || COUNT(*) FROM disposal;

-- STEP 1: SUCCESSFUL TWO-PHASE COMMIT
BEGIN;
```

Data Output | Messages | Explain | Notifications

Showing rows: 1 to 1

	?column? text
1	Initial Disposal count...

```sql
-- Phase 2: Commit both
COMMIT;

DO $$
BEGIN
    RAISE NOTICE '=== SUCCESSFUL 2PC COMPLETED ===';
    RAISE NOTICE 'Collection_A after commit: %', (SELECT COUNT(*) FROM collection_a);
    RAISE NOTICE 'Disposal after commit: %', (SELECT COUNT(*) FROM disposal);
END $$;
```

Data Output | Messages | Explain | Notifications

```
NOTICE:  === SUCCESSFUL 2PC COMPLETED ===
NOTICE:  Collection_A after commit: 6
NOTICE:  Disposal after commit: 1
DO

Query returned successfully in 59 msec.
```

Step 3: Simulate Failure & In-Doubt Transaction

```
EXCEPTION
    WHEN others THEN
        ROLLBACK TO SAVEPOINT before_remote_insert;
        -- Transaction is now in "in-doubt" state
        RAISE NOTICE 'Transaction failed at remote operation. Manual intervention required.';
        ROLLBACK;
```

Data Output | Messages | Explain ✕ | Notifications

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

Query returned successfully in 39 msec.
```

Step 4: Check Transaction State

```
--Step 4: Check Transaction State
-- Check for any prepared transactions
SELECT * FROM pg_prepared_xacts;

-- Check current locks that might indicate stuck transact
SELECT
    locktype,
    relation::regclass,
    mode,
    granted
FROM pg_locks
```

Data Output | Messages | Explain ✕ | Notifications

	transaction xid	gid text	prepared timestamp with time zone	owner name	database name

```sql
-- Check current locks that might indicate stuck transactions
SELECT
    locktype,
    relation::regclass,
    mode,
    granted
FROM pg_locks
WHERE NOT granted;

-- Verify no data was committed from failed transaction
SELECT 'Collection_A count after failure: ' || COUNT(*) FROM collection_a
WHERE collection_id = 12;

SELECT 'Disposal count after failure: ' || COUNT(*) FROM disposal
WHERE collection_id = 999;
```

Data Output | Messages | Explain | Notifications

	locktype text	relation regclass	mode text	granted boolean

```sql
-- Verify no data was committed from failed transaction
SELECT 'Collection_A count after failure: ' || COUNT(*) FROM collection_A
WHERE collection_id = 12;

SELECT 'Disposal count after failure: ' || COUNT(*) FROM disposal
WHERE collection_id = 999;
```

Data Output | Messages | Explain | Notifications

Showing rows: 1 to 1 | Page No: 1

	?column? text
1	Collection_A count after failure...

Step 5: Manual Recovery & Force Operations

```sql
-- Verify total committed rows remain ≤10 in main collection table
SELECT 'Total committed rows in collection_a: ' || COUNT(*)
FROM collection_A
WHERE collection_id <= 13;
```

Data Output | Messages | Explain | Notifications

Showing rows: 1 to 1 Page No: 1

	?column? text
1	Total committed rows in collection_...

```sql
DO $$
BEGIN
    BEGIN
        INSERT INTO collection_a VALUES (12, 112, 202, CURRENT_DATE, 21.3, 'PAPER', 'COMPLETED');
        INSERT INTO disposal VALUES (DEFAULT, 12, CURRENT_DATE, 'LANDFILL', 'City Dump');
        -- Simulate failure before commit
        RAISE EXCEPTION 'NETWORK_FAILURE: Connection lost between nodes';
        COMMIT;
    EXCEPTION
        WHEN others THEN
            RAISE NOTICE 'Transaction failed: %', SQLERRM;
            ROLLBACK;
    END;
END $$;
```

Data Output | Messages | Explain | Notifications

```
NOTICE:  Transaction failed: NETWORK_FAILURE: Connection lost between nodes
DO

Query returned successfully in 42 msec.
```

```sql
-- Verify one-to-one relationship
SELECT 'Consistency check - matched pairs: ' || COUNT(*)
FROM collection_a c
JOIN disposal d ON c.collection_id = d.collection_id
WHERE c.collection_id IN (11, 13);
```

Data Output | Messages | Explain | Notifications

Showing rows: 1 to 1 Page No: 1

	?column? text
1	Consistency check - matched pair...

```sql
--Step 1: Prepare Test Data
--ensure we have the right data for the lock conflict:

-- Verify we have collection_id 11 from previous A4 test
SELECT collection_id, waste_type, weight_kg
FROM collection_a
WHERE collection_id = 11;
```

Output Messages Explain ✕ Notifications

Showir

collection_id [PK] integer	waste_type character varying (20)	weight_kg numeric (8,2)
11	PLASTIC	17.50

```sql
-- Verify the lock is held
SELECT 'Session 1: Lock acquired on collection_id 11 - transaction open';
SELECT pg_sleep(1);  -- Keep session alive

-- Keep this transaction OPEN - DO NOT COMMIT YET
```

Data Output Messages Explain ✕ Notifications

Showing rows: 1 to 1 Page No: 1

	pg_sleep void
1	

Step 4: Lock Diagnostics

```sql
--Step 4: Lock Diagnostics

-- Check for blocking locks
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,
    blocked_activity.application_name AS blocked_application,
    blocking_activity.application_name AS blocking_application
FROM pg_catalog.pg_locks blocked_locks
JOIN pg_catalog.pg_stat_activity blocked_activity ON blocked_activity.pid = blocked_locks.pid
JOIN pg_catalog.pg_locks blocking_locks ON blocking_locks.locktype = blocked_locks.locktype
```

Data Output Messages Explain Notifications

	blocked_pid integer	blocked_user name	blocking_pid integer	blocking_user name	blocked_statement text	current_statement_in_blocking_process text	blocked_application text	blocking_a text

```sql
-- Check current transactions

SELECT

    usename,
    application_name,
    state,
    query,
    now() - query_start AS duration
FROM pg_stat_activity
WHERE state != 'idle'
ORDER BY duration DESC;
```

Data Output Messages Explain Notifications

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

	usename name	application_name text	state text	query text
1	postgres	pgAdmin 4 - CONN:45500...	active	SELECT

```
--Step 5: Timestamp Evidence
-- Record timestamps before lock release
SELECT 'Time before lock release: ' || now();

-- Show that Session 2 is still waiting
SELECT 'Session 2 state: ' || state
FROM pg_stat_activity
WHERE query LIKE '%UPDATE collection_a%'
AND state = 'active';
```

Data Output | Messages | Explain | Notifications

Showing rows: 1 to 1 Page No: 1

	?column? text
1	Time before lock release: 2025-10-29 00:07:29.73463...

```
-- Show that Session 2 is still waiting
SELECT 'Session 2 state: ' || state
FROM pg_stat_activity
WHERE query LIKE '%UPDATE collection_a%'
AND state = 'active';
```

Data Output | Messages | Explain | Notifications

Showing rows: 1 to 1

	?column? text
1	Session 2 state: acti...

```sql

--Step 6: Release Lock & Verify Completion
--check for session 1

-- SESSION 1: Commit to release the lock
COMMIT;

-- Verify the update
SELECT collection_id, weight_kg
FROM collection_a
WHERE collection_id = 11;
```

Data Output Messages Explain ✕ Notifications

Showing rows: 1 to 1

	collection_id [PK] integer	weight_kg numeric (8,2)
1	11	25.00

```sql

--check Session 2
-- Session 2 should now show COMMIT completed
SELECT 'Session 2 completed at: ' || now();

-- Verify final state
SELECT collection_id, weight_kg
FROM collection_a
WHERE collection_id = 11;
```

Data Output Messages Explain ✕ Notifications

Showing rows: 1 to 1

	collection_id [PK] integer	weight_kg numeric (8,2)
1	11	25.00