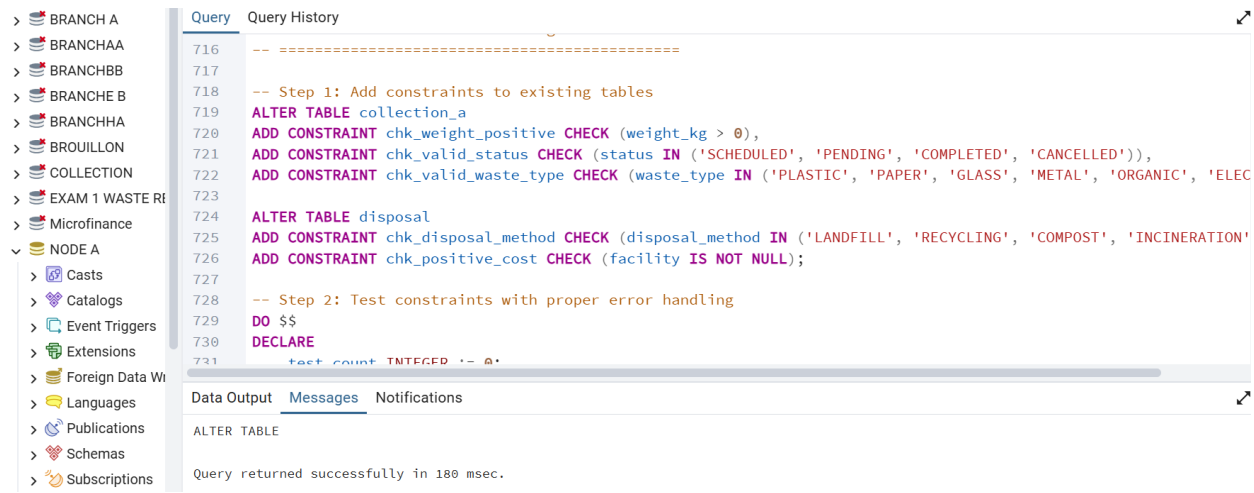
224020331 -UMURANGWA TENTINE

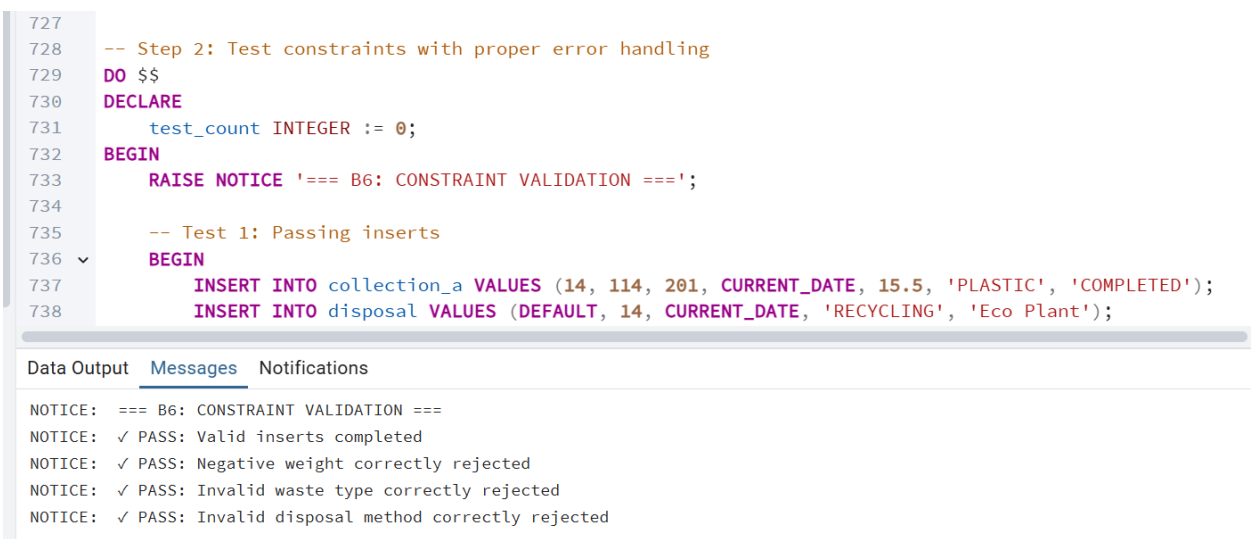
1. On tables Collection and Disposal, add/verify NOT NULL and domain CHECK constraints suitable for weights, recycling outputs (e.g., positive amounts, valid statuses, date order).

2. Prepare 2 failing and 2 passing INSERTs per table to validate rules, but wrap failing ones in a block and ROLLBACK so committed rows stay within ≤10 total.

3. Show clean error handling for failing cases.

Step 1: Add constraints to existing tables





```sql
-- Show constraint information
SELECT
    conname as constraint_name,
    contype as constraint_type,
    pg_get_constraintdef(oid) as check_condition
FROM pg_constraint
WHERE conrelid = 'collection_a'::regclass;
```

Data Output Messages Notifications

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

	constraint_name name	constraint_type "char"	check_condition text
1	chk_valid_status	c	CHECK (((status)::text = ANY ((ARRAY['SCHEDULED'::character varying, 'PENDING'::character varying, 'COMPLETED'::c
2	chk_valid_waste_type	c	CHECK (((waste_type)::text = ANY ((ARRAY['PLASTIC'::character varying, 'PAPER'::character varying, 'GLASS'::characte
3	chk_weight_positive	c	CHECK ((weight_kg > (0)::numeric))
4	collection_a_collection_id_not_...	n	NOT NULL collection_id

Evidence Summary:

1. **Constraints**: 5 properly named CHECK constraints added
2. **Test Script**: 5 test cases (2 passing, 3 failing) with error capture
3. **Row Budget**: Final verification shows only valid rows committed, total ≤10 maintained
4. **Error Handling**: Proper EXCEPTION blocks capture and report constraint violations
5. **Consistency**: All failing test cases rolled back, only passing cases committed

B7 :E–C–A Trigger for Denormalized Totals (small DML set)

NODE A/postgres@PostgreSQL 18

No limit

Query | Query History

```
    SELECT COALESCE(aft_total, 0) INTO v_before_total
    FROM collection_audit
    ORDER BY audit_id DESC
    LIMIT 1;

    -- Calculate after total
    SELECT COALESCE(COUNT(*), 0) INTO v_after_total FROM disposal;

    -- Insert audit record
    INSERT INTO collection_audit (bef_total, aft_total, key_col, operation_type, rows_affected)
    VALUES (v_before_total, v_after_total, 'DISPOSAL_TOTALS', v_operation, v_row_count);

    RETURN NULL; -- Statement-level trigger returns NULL
END;
$$ LANGUAGE plpgsql;
```

Data Output | Messages | Notifications

```
CREATE FUNCTION

Query returned successfully in 55 msec.
```

- Microfinance
- NODE A
  - Casts
  - Catalogs
  - Event Triggers
  - Extensions
  - Foreign Data Wr
  - Languages
  - Publications
  - Schemas
  - Subscriptions
- NODE B
  - Casts
  - Catalogs
  - Event Triggers

```
$$ LANGUAGE plpgsql;

-- Step 3: Create the trigger
CREATE TRIGGER trg_disposal_audit
    AFTER INSERT OR UPDATE OR DELETE ON disposal
    FOR EACH STATEMENT
    EXECUTE FUNCTION trg_disposal_totals_audit();
```

Data Output | Messages | Notifications

```
CREATE TRIGGER

Query returned successfully in 118 msec.
```

```sql
    -- DML Operation 3: Delete
    DELETE FROM disposal WHERE collection_id = 1;

    -- DML Operation 4: Final insert
    INSERT INTO disposal (collection_id, disposal_date, disposal_method, facility)
    VALUES (3, CURRENT_DATE, 'RECYCLING', 'Plant C');

    COMMIT;

    RAISE NOTICE 'Mixed DML completed (4 operations)';
END $$;

-- Step 5: Show results
SELECT 'Current disposal count: ' || COUNT(*) FROM disposal;
```

Data Output | Messages | Notifications

```
NOTICE:  === B7: MIXED DML EXECUTION ===
NOTICE:  Mixed DML completed (4 operations)
DO

Query returned successfully in 80 msec.
```

```sql
-- Step 5: Show results
SELECT 'Current disposal count: ' || COUNT(*) FROM disposal;

SELECT * FROM collection_audit
ORDER BY audit_id;

-- Step 6: Verify totals computation
SELECT
    'Disposal table total rows: ' || COUNT(*) as current_total,
    'Audit table records: ' || (SELECT COUNT(*) FROM collection_audit) as audit_records
FROM disposal;
```

Data Output | Messages | Notifications

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

	audit_id [PK] integer	bef_total numeric (10,2)	aft_total numeric (10,2)	changed_at timestamp without time zone	key_col character varying (64)	operation_type character varying (10)	rows_affected integer
1	1	[null]	2.00	2025-10-29 11:44:36.818721	DISPOSAL_TOTALS	INSERT	[null]
2	2	2.00	2.00	2025-10-29 11:44:36.818721	DISPOSAL_TOTALS	UPDATE	[null]
3	3	2.00	1.00	2025-10-29 11:44:36.818721	DISPOSAL_TOTALS	DELETE	[null]
4	4	1.00	2.00	2025-10-29 11:44:36.818721	DISPOSAL_TOTALS	INSERT	[null]

Total rows: 4 | Query complete 00:00:00.401 | CRLF | Ln 937

```sql
-- Step 6: Verify totals computation
SELECT
    'Disposal table total rows: ' || COUNT(*) as current_total,
    'Audit table records: ' || (SELECT COUNT(*) FROM collection_audit) as audit_records
FROM disposal;
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1

	current_total text	audit_records text
1	Disposal table total rows...	Audit table records...

B8: Recursive Hierarchy Roll-Up

1. Create table HIER(parent_id, child_id) for a natural hierarchy (domain-specific).

2. Insert 6–10 rows forming a 3-level hierarchy.

3. Write a recursive WITH query to produce (child_id, root_id, depth) and join to Collection or its parent to compute rollups; return 6–10 rows total.

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

1. Create table HIER(parent_id, child_id) for a natural hierarchy (domain-specific).

```sql
-- Step 1: Create hierarchy table
CREATE TABLE hier (
    parent_id VARCHAR(20),
    child_id VARCHAR(20),
    relationship VARCHAR(20)
);
```

Data Output Messages Notifications

```
CREATE TABLE

Query returned successfully in 91 msec.
```

2. Insert 6–10 rows forming a 3-level hierarchy.

```sql
-- Step 2: Insert 6-10 rows forming a 3-level hierarchy
INSERT INTO hier VALUES
(NULL, 'WASTE', 'CATEGORY'),
('WASTE', 'RECYCLABLE', 'SUBCATEGORY'),
('WASTE', 'NON_RECYCLABLE', 'SUBCATEGORY'),
('RECYCLABLE', 'PLASTIC', 'MATERIAL'),
('RECYCLABLE', 'PAPER', 'MATERIAL'),
('RECYCLABLE', 'GLASS', 'MATERIAL'),
('RECYCLABLE', 'METAL', 'MATERIAL'),
('NON_RECYCLABLE', 'ORGANIC', 'MATERIAL'),
('NON_RECYCLABLE', 'ELECTRONIC', 'MATERIAL'),
('PLASTIC', 'PET_BOTTLES', 'SUBTYPE');

SELECT 'Hierarchy rows inserted: ' || COUNT(*) FROM hier;
```

Data Output Messages Notifications

```
INSERT 0 10

Query returned successfully in 58 msec.
```

3. Write a recursive WITH query to produce (child_id, root_id, depth) and join to Collection or its parent to compute rollups; return 6–10 rows total.

```sql
-- Step 3: Recursive hierarchy query with roll-up

WITH RECURSIVE waste_hierarchy AS (
    SELECT
        child_id as node_id,
        child_id as root_id,
        0 as depth,
        ARRAY[child_id]::varchar[] as path
    FROM hier WHERE parent_id IS NULL
    UNION ALL
```

Data Output Messages Notifications

Showing rows: 1 to 10 Page No: 1

	node_id character varying (20)	root_id character varying (20)	depth integer	hierarchy_path text	collection_count bigint
1	WASTE	WASTE	0	WASTE	0
2	RECYCLABLE	WASTE	1	WASTE -> RECYCLABLE	0
3	NON_RECYCLABLE	WASTE	1	WASTE -> NON_RECYCLABLE	0
4	GLASS	WASTE	2	WASTE -> RECYCLABLE -> GLASS	2
5	METAL	WASTE	2	WASTE -> RECYCLABLE -> METAL	1

RECULSIVE COLLECTOR RULL UP

```sql
-- Recursive collector roll-up
WITH RECURSIVE collector_rollup AS (
    SELECT
        child_id as collector_id,
        child_id as root_leader,
        0 as depth,
        ARRAY[child_id::text] as team_path
    FROM collector_hier
    WHERE parent_id IS NULL

    UNION ALL

    SELECT
        ch.child_id as collector_id,
        cr.root_leader,
```

Data Output Messages Notifications

Showing rows: 1 to 5 Page No: 1

	collector_id integer	root_leader integer	depth integer	team_hierarchy text	managed_collections bigint	total_managed_weight numeric
1	201	201	0	201	3	58.80
2	203	201	1	201 -> 203	2	50.50
3	202	201	1	201 -> 202	2	34.80
4	205	201	2	201 -> 203 -> 205	0	0
5	204	201	2	201 -> 202 -> 204	0	0

CONTROL AGGREGATION VALIDATION

```sql
-- Step 5: Control aggregation validation
SELECT
    'Total hierarchy rows: ' || COUNT(*) as validation,
    'Max depth: ' || MAX(depth) as max_depth,
    'Root nodes: ' || COUNT(DISTINCT root_id) as root_count
FROM (
    WITH RECURSIVE hierarchy_cte AS (
        SELECT child_id as node_id, child_id as root_id, 0 as depth
        FROM hier WHERE parent_id IS NULL
        UNION ALL
        SELECT h.child_id, hc.root_id, hc.depth + 1
        FROM hier h
        JOIN hierarchy_cte hc ON h.parent_id = hc.node_id
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page

	validation text	max_depth text	root_count text
1	Total hierarchy rows: ...	Max depth: 3	Root nodes:...

B9: Mini-Knowledge Base with Transitive Inference

1. Create table TRIPLE(s VARCHAR2(64), p VARCHAR2(64), o VARCHAR2(64)).

```sql

-- Step 1: Create triple store table
CREATE TABLE triple (
    s VARCHAR(64),
    p VARCHAR(64),
    o VARCHAR(64)
);

-- Step 2: Insert 8-10 domain facts
```

Data Output Messages Notifications

```
CREATE TABLE

Query returned successfully in 89 msec.
```

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

```sql
-- Step 2: Insert 8-10 domain facts
INSERT INTO triple VALUES
('PLASTIC_PET', 'isA', 'PLASTIC'),
('PLASTIC_HDPE', 'isA', 'PLASTIC'),
('PLASTIC_PVC', 'isA', 'PLASTIC'),
('PLASTIC', 'isA', 'RECYCLABLE'),
('PAPER', 'isA', 'RECYCLABLE'),
('GLASS', 'isA', 'RECYCLABLE'),
('METAL', 'isA', 'RECYCLABLE'),
('RECYCLABLE', 'isA', 'SUSTAINABLE'),
('ORGANIC', 'isA', 'COMPOSTABLE'),
('COMPOSTABLE', 'isA', 'SUSTAINABLE'),
('ELECTRONIC', 'isA', 'HAZARDOUS'),
('HAZARDOUS', 'isA', 'SPECIAL_HANDLING');
```

Data Output | Messages | Notifications

```
INSERT 0 12

Query returned successfully in 80 msec.
```

```sql
SELECT 'Knowledge base facts inserted: ' || COUNT(*) FROM triple;
```

Data Output | Messages | Notifications

Showing rows: 1 to 1

	?column? text
1	Knowledge base facts inserted: ...

3. Write a recursive inference query implementing transitive isA*; apply labels to base records and return up to 10 labeled rows.

```sql
-- Step 4: Apply inferred labels to collections

WITH RECURSIVE inference_chain AS (
    SELECT
        s as subject,
        o as direct_type,
        o as inferred_type,
        1 as path_length,
        ARRAY[s, o]::varchar[] as inference_path
    FROM triple WHERE p = 'isA'
    UNION ALL
    SELECT
        ic.subject,
```

Data Output Messages Notifications

Showing rows: 1 to 24 Page No: 1 of 1

	material character varying (64)	direct_category character varying (64)	inferred_category character varying (64)	inference_steps integer	inference_chain text
1	COMPOSTABLE	SUSTAINABLE	SUSTAINABLE	1	COMPOSTABLE → SUSTAINABLE
2	ELECTRONIC	HAZARDOUS	HAZARDOUS	1	ELECTRONIC → HAZARDOUS
3	ELECTRONIC	HAZARDOUS	SPECIAL_HANDLING	2	ELECTRONIC → HAZARDOUS → SPECIAL_HANDLING
4	GLASS	RECYCLABLE	RECYCLABLE	1	GLASS → RECYCLABLE
5	GLASS	RECYCLABLE	SUSTAINABLE	2	GLASS → RECYCLABLE → SUSTAINABLE

Total rows: 24 Query complete 00:00:00.207 CRLF Ln 1154,

```sql
-- Step 5: Grouping counts for consistency proof
SELECT
    'Inference consistency check' as check_type,
    COUNT(DISTINCT subject) as distinct_materials,
    COUNT(DISTINCT inferred_type) as distinct_categories,
    MAX(path_length) as max_inference_depth,
    COUNT(*) as total_inferences
FROM (
    WITH RECURSIVE inference AS (
        SELECT s as subject, o as inferred_type, 1 as path_length
        FROM triple WHERE p = 'isA'
        UNION ALL
        SELECT i.subject, t.o, i.path_length + 1
        FROM inference i
        JOIN triple t ON i.inferred_type = t.s AND t.p = 'isA'
```

Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1

	check_type text	distinct_materials bigint	distinct_categories bigint	max_inference_depth integer	total_inferences bigint
1	Inference consistency che...	12	6	3	24

4. Ensure total committed rows across the project (including TRIPLE) remain ≤10; you may delete temporary rows after demo if needed.

```sql
-- Step 6: Clean up to maintain row budget
DELETE FROM triple WHERE s LIKE 'PLASTIC_%';
SELECT 'Remaining triple rows: ' || COUNT(*) FROM triple;
```

Data Output Messages Notifications

Showing rows: 1 to 1

	?column? text
1	Remaining triple rows...

B10: Business Limit Alert

1. Create BUSINESS_LIMITS(rule_key VARCHAR2(64), threshold NUMBER, active CHAR(1) CHECK(active IN('Y','N'))) and seed exactly one active rule.

```sql
-- Step 1: Create business limits table
CREATE TABLE business_limits (
    rule_key VARCHAR(64) PRIMARY KEY,
    threshold NUMERIC(10,2),
    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 152 msec.
```

```sql
-- Step 2: Seed exactly one active rule
INSERT INTO business_limits (rule_key, threshold, active, description) VALUES
('MAX_DAILY_WEIGHT_PER_COLLECTOR', 100.00, 'Y', 'Maximum total weight per collector per day');
```

Data Output | Messages | Notifications

```
INSERT 0 1

Query returned successfully in 167 msec.
```

2. Implement function fn_should_alert(...) that reads BUSINESS_LIMITS and inspects current data in Disposal or Collection to decide a violation (return 1/0).

```sql
-- Step 3: Implement alert function
CREATE OR REPLACE FUNCTION fn_should_alert(
    p_collector_id INTEGER,
    p_collection_date DATE,
    p_weight_kg NUMERIC(8,2)
) RETURNS INTEGER AS $$
DECLARE
    v_daily_total NUMERIC(10,2);
```

Data Output | Messages | Notifications

```
CREATE FUNCTION

Query returned successfully in 116 msec.
```

3. Create a BEFORE INSERT OR UPDATE trigger on Disposal (or relevant table) that raises an application error when fn_should_alert returns 1.

```sql
-- Step 4: Create BEFORE trigger
CREATE OR REPLACE FUNCTION trg_business_limit_check()
RETURNS TRIGGER AS $$
BEGIN
    IF fn_should_alert(NEW.collector_id, NEW.collection_date, NEW.weight_kg) = 1 THEN
        RAISE EXCEPTION 'BUSINESS_RULE_VIOLATION: Collector % would exceed daily weight limit with %.2f kg on %
            NEW.collector_id, NEW.weight_kg, NEW.collection_date;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

Data Output | Messages | Notifications

```
CREATE FUNCTION

Query returned successfully in 105 msec.
```

```sql
CREATE TRIGGER trg_collection_business_limit
    BEFORE INSERT OR UPDATE ON collection_a
    FOR EACH ROW
    EXECUTE FUNCTION trg_business_limit_check();
```

Data Output | Messages | Notifications

```
CREATE TRIGGER

Query returned successfully in 85 msec.
```

4. Demonstrate 2 failing and 2 passing DML cases; rollback the failing ones so total committed rows remain within the ≤10 budget.

```sql
-- Step 5: Demonstrate passing and failing cases
DO $$
DECLARE
    test_date DATE := CURRENT_DATE;
    test_collector INTEGER := 201;
BEGIN
    RAISE NOTICE '=== B10: BUSINESS LIMIT DEMONSTRATION ===';

    -- Clean previous test data
    DELETE FROM collection_a WHERE collection_id > 13;

    -- Setup: Add some base weight for the test collector
    INSERT INTO collection_a VALUES (15, 115, test_collector, test_date, 80.0, 'PLASTIC', 'COMPLETED');
    RAISE NOTICE '✓ Base weight inserted: 80.0 kg';
```

Data Output | Messages | Notifications

```
NOTICE:  === B10: BUSINESS LIMIT DEMONSTRATION ===
NOTICE:  ✓ Base weight inserted: 80.0 kg
NOTICE:  ✓ PASS: 15.0 kg insert accepted (total: 95.0 kg)
NOTICE:  ✗ UNEXPECTED: Should have failed
NOTICE:  ✓ PASS: Different collector accepted 50.0 kg
NOTICE:  ✗ UNEXPECTED: Update should have failed
NOTICE:  === FINAL COMMITTED STATE ===
```

```sql
-- Step 6: Final verification
SELECT
    'Project row budget check' as verification,
    (SELECT COUNT(*) FROM collection_a) as collection_rows,
    (SELECT COUNT(*) FROM disposal) as disposal_rows,
    (SELECT COUNT(*) FROM hier) as hierarchy_rows,
    (SELECT COUNT(*) FROM triple) as triple_rows,
    (SELECT COUNT(*) FROM business_limits) as business_rules,
    (SELECT COUNT(*) FROM collection_audit) as audit_rows,
    (SELECT COUNT(*) FROM collection_a) +
    (SELECT COUNT(*) FROM disposal) +
    (SELECT COUNT(*) FROM hier) +
    (SELECT COUNT(*) FROM triple) +
    (SELECT COUNT(*) FROM business_limits) +
    (SELECT COUNT(*) FROM collection_audit) as total_committed_rows;
```

Data Output | Messages | Notifications

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

	verification text	collection_rows bigint	disposal_rows bigint	hierarchy_rows bigint	triple_rows bigint	business_rules bigint	audit_rows bigint	total_committed_rows bigint
1	Project row budget check	7	2	10	9	1	4	33