FAO Database Usage

PostgreSQL Rules & Gotchas

1. Column Alias Restrictions

```sql
-- ❌ WRONG - Can't use alias in WHERE/ORDER BY
SELECT value * 2 as doubled FROM table ORDER BY doubled;

-- ✅ CORRECT - Use column position or repeat expression
SELECT value * 2 as doubled FROM table ORDER BY 1;
SELECT value * 2 as doubled FROM table ORDER BY value * 2;
```

2. ROUND Requires Numeric Cast

```sql
-- ❌ WRONG
SELECT ROUND(value1 / value2, 2) FROM table;

-- ✅ CORRECT
SELECT ROUND((value1 / value2)::numeric, 2) FROM table;
```

3. NaN and NULL Handling

```sql
-- Check for valid numeric values
WHERE value IS NOT NULL
  AND value != 'NaN'::float
  AND value > 0
```

Core Exploration Principles

1. Never Assume - Always Discover

Start with metadata queries:

```sql
-- What elements exist in this table?
SELECT DISTINCT
    e.element,
    e.element_code,
    COUNT(*) as usage_count
FROM production_crops_livestock pcl
JOIN elements e ON e.id = pcl.element_code_id
GROUP BY e.element, e.element_code
ORDER BY usage_count DESC;

-- What items are in the database?
SELECT
    ic.item,
    COUNT(DISTINCT pcl.year) as years_available,
    MIN(pcl.year) as first_year,
    MAX(pcl.year) as last_year
FROM production_crops_livestock pcl
JOIN item_codes ic ON ic.id = pcl.item_code_id
GROUP BY ic.item
ORDER BY years_available DESC
LIMIT 50;
```

2. Use Pattern Matching for Discovery

Instead of hardcoding values, use LIKE:

```sql
-- Find all wheat varieties
SELECT DISTINCT item
FROM item_codes
WHERE LOWER(item) LIKE '%wheat%';

-- Find all production-related elements
SELECT DISTINCT element
FROM elements
WHERE LOWER(element) LIKE '%production%'
   OR LOWER(element) LIKE '%yield%'
    OR LOWER(element) LIKE '%harvest%';
```

3. Understand Data Quality Through Flags

Always check flag distribution:

```sql
-- Flag analysis for any table
SELECT
    f.flag,
    f.description,
    COUNT(*) as records,
    ROUND((COUNT(*) * 100.0 / SUM(COUNT(*)) OVER())::numeric, 2) as percentage
FROM production_crops_livestock pcl
JOIN flags f ON f.id = pcl.flag_id
WHERE pcl.year >= 2020
GROUP BY f.flag, f.description
ORDER BY records DESC;
```

Key flags to remember:

- (A) = Official figure (most reliable)
- (X) = Figure from international organizations
- (E) = Estimated value
- (P) = Provisional value
- (M) = Missing value (data cannot exist)
- (O) = Missing value
- (U) = Low reliability

Effective Exploration Queries

1. Data Coverage Analysis

sql

```sql
-- Which countries have the most complete data?
WITH country_coverage AS (
    SELECT
        ac.area,
        COUNT(DISTINCT pcl.year) as years_with_data,
        COUNT(DISTINCT ic.item) as items_tracked,
        SUM(CASE WHEN f.flag = 'A' THEN 1 ELSE 0 END) as official_records,
        COUNT(*) as total_records
    FROM production_crops_livestock pcl
    JOIN area_codes ac ON ac.id = pcl.area_code_id
    JOIN item_codes ic ON ic.id = pcl.item_code_id
    JOIN flags f ON f.id = pcl.flag_id
    WHERE pcl.year >= 2015
        AND pcl.value > 0
    GROUP BY ac.area
)
SELECT
    area,
    years_with_data,
    items_tracked,
    ROUND((official_records * 100.0 / total_records)::numeric, 2) as pct_official
FROM country_coverage
ORDER BY years_with_data DESC, items_tracked DESC
LIMIT 30;
```

2. Find Interesting Patterns

sql

```sql
-- Discover unusual year-over-year changes
WITH value_changes AS (
  SELECT
    ic.item,
    ac.area,
    pcl.year,
    pcl.value,
    LAG(pcl.value) OVER (PARTITION BY ic.item, ac.area ORDER BY pcl.year) as prev_value,
    ec.element
  FROM production_crops_livestock pcl
  JOIN item_codes ic ON ic.id = pcl.item_code_id
  JOIN area_codes ac ON ac.id = pcl.area_code_id
  JOIN elements ec ON ec.id = pcl.element_code_id
  JOIN flags f ON f.id = pcl.flag_id
  WHERE pcl.value > 0
    AND f.flag IN ('A', 'X', 'E')
    AND pcl.year BETWEEN 2018 AND 2023
)
SELECT
  item,
  area,
  year,
  element,
  value,
  prev_value,
  ROUND(((value - prev_value) / NULLIF(prev_value, 0) * 100)::numeric, 2) as pct_change
FROM value_changes
WHERE prev_value > 0
  AND ABS(value - prev_value) / prev_value > 0.3  -- 30% change threshold
ORDER BY ABS(value - prev_value) / prev_value DESC
LIMIT 50;
```

3. Cross-Table Exploration

sql

```sql
-- Compare production vs trade for discovery
WITH production_summary AS (
  SELECT
    ic.item,
    SUM(pcl.value) as total_production,
    COUNT(DISTINCT ac.area) as producing_countries
  FROM production_crops_livestock pcl
  JOIN item_codes ic ON ic.id = pcl.item_code_id
  JOIN area_codes ac ON ac.id = pcl.area_code_id
  JOIN elements ec ON ec.id = pcl.element_code_id
  WHERE LOWER(ec.element) LIKE '%production%'
    AND pcl.year = 2022
    AND pcl.value > 0
  GROUP BY ic.item
),
trade_summary AS (
  SELECT
    ic.item,
    SUM(CASE WHEN LOWER(ec.element) LIKE '%export%' THEN tcl.value ELSE 0 END) as total_exports
    SUM(CASE WHEN LOWER(ec.element) LIKE '%import%' THEN tcl.value ELSE 0 END) as total_imports
  FROM trade_crops_livestock tcl
  JOIN item_codes ic ON ic.id = tcl.item_code_id
  JOIN elements ec ON ec.id = tcl.element_code_id
  WHERE tcl.year = 2022
    AND tcl.value > 0
  GROUP BY ic.item
)
SELECT
  p.item,
  p.total_production,
  p.producing_countries,
  t.total_exports,
  t.total_imports,
  ROUND((t.total_exports / NULLIF(p.total_production, 0) * 100)::numeric, 2) as export_ratio_pc
FROM production_summary p
LEFT JOIN trade_summary t ON p.item = t.item
WHERE p.total_production > 1000000  -- Focus on major crops
ORDER BY p.total_production DESC
LIMIT 30;
```

Tips for Filtering Countries vs Regions

Understanding area_code_m49

```sql
-- Explore the M49 code patterns
SELECT
  area,
  area_code,
  area_code_m49,
  LENGTH(area_code_m49) as m49_length
FROM area_codes
WHERE area_code_m49 IS NOT NULL
ORDER BY m49_length, area;
```

Common patterns:

- Individual countries: Usually 3-digit codes (but verify!)
- Regions: Various lengths
- Better approach: Create a view of confirmed countries

```sql
-- Create a reliable country filter
CREATE VIEW individual_countries AS
SELECT DISTINCT area_code, area, area_code_m49
FROM area_codes
WHERE area NOT LIKE '%World%'
  AND area NOT LIKE '% region%'
  AND area NOT LIKE '% countries%'
  AND area NOT LIKE '%European Union%'
  AND area NOT IN (
    'Africa', 'Americas', 'Asia', 'Europe', 'Oceania',
    'Northern Africa', 'Sub-Saharan Africa',
    'Eastern Asia', 'Southern Asia', 'South-Eastern Asia', 'Western Asia',
    'Eastern Europe', 'Northern Europe', 'Southern Europe', 'Western Europe',
    'Caribbean', 'Central America', 'South America', 'Northern America'
  );
```

Element Discovery Patterns

```sql
-- Find all unique element patterns in a table
SELECT
    element,
    COUNT(*) as usage_count,
    STRING_AGG(DISTINCT pcl.unit, ', ') as units_used
FROM production_crops_livestock pcl
JOIN elements e ON e.id = pcl.element_code_id
GROUP BY element
ORDER BY usage_count DESC;

-- Discover element relationships
SELECT
    e1.element as element1,
    e2.element as element2,
    COUNT(*) as paired_occurrences
FROM production_crops_livestock p1
JOIN production_crops_livestock p2
    ON p1.area_code_id = p2.area_code_id
    AND p1.item_code_id = p2.item_code_id
    AND p1.year = p2.year
JOIN elements e1 ON e1.id = p1.element_code_id
JOIN elements e2 ON e2.id = p2.element_code_id
WHERE e1.element < e2.element  -- Avoid duplicates
    AND p1.value > 0 AND p2.value > 0
GROUP BY e1.element, e2.element
HAVING COUNT(*) > 1000
ORDER BY paired_occurrences DESC;
```

Best Practices Summary

1. **Always explore first** - Don't assume table contents
2. **Use flags for quality** - Filter by A, X, E for reliable data
3. **Pattern match with LIKE** - More flexible than exact matches
4. **Check for NaN/NULL** - Both exist in FAO data
5. **Cast for calculations** - PostgreSQL is strict about types
6. **Join on IDs** - Always use foreign key relationships
7. **Verify assumptions** - What works in one table might not in another

Template for New Table Exploration

sql

```sql
-- 1. Basic structure
SELECT column_name, data_type
FROM information_schema.columns
WHERE table_name = 'YOUR_TABLE_NAME';

-- 2. Row count and date range
SELECT
  COUNT(*) as total_rows,
  MIN(year) as earliest_year,
  MAX(year) as latest_year
FROM YOUR_TABLE_NAME;

-- 3. Flag distribution
SELECT f.flag, f.description, COUNT(*) as count
FROM YOUR_TABLE_NAME t
JOIN flags f ON f.id = t.flag_id
GROUP BY f.flag, f.description
ORDER BY count DESC;

-- 4. Sample data
SELECT * FROM YOUR_TABLE_NAME
WHERE year >= 2020
LIMIT 20;
```

Remember: The goal is to understand the data's structure and quality before making any analytical queries!