DTS207TC: Database Development and Design

***Lab Manual**: Data Warehouse*

Supervisor

Dr. Di Zhang

Dr. Hengyan Liu

School of AI and Advanced Computing

Xi'an Jiaotong-Liverpool University

November 2025

1 Introduction

1.1 Concept Recap

Definition: A **data warehouse** stores integrated, cleaned, and historical data optimized for analysis and decision support. It is predominantly *read-heavy* and supports strategic insights, unlike OLTP systems that manage day-to-day operational transactions. *(Ref: Lec7 — Data Warehouse)*

OLAP model:

- **Fact table:** Contains quantitative measures such as `sales_amount`, linked to multiple dimensions.
- **Dimension tables:** Provide descriptive context such as `product`, `time`, and `region` — the "nouns" of analytics.

(Ref: Lec7 — Data Warehouse)

Common physical model:

- **Star schema:** A single central fact table surrounded by dimension tables.

(Ref: Lec7 — Data Warehouse)

We will build:

- `dim_date`
- `dim_product`
- `dim_region`
- `fact_sales`

1.2 Create Schema

1.2.1 Create Dimension Tables

Listing 1: Creating dimension tables

```
CREATE TABLE dim_date (
    date_key        DATE PRIMARY KEY,        -- Surrogate or natural key for date
    year            INT,
    quarter         TEXT,                    -- e.g. 'Q1', 'Q2', ...
    month_num       INT,
    month_name      TEXT
);

CREATE TABLE dim_product (
    product_key     SERIAL PRIMARY KEY,
    product_name    TEXT,
    category        TEXT,
    subcategory     TEXT
);

CREATE TABLE dim_region (
    region_key      SERIAL PRIMARY KEY,
    region_name     TEXT,            -- e.g. 'North America'
    country         TEXT,            -- e.g. 'USA'
    city            TEXT             -- e.g. 'New York'
);
```

1.2.2 Create Fact Table

Listing 2: Creating the fact$_s alestable$

```
CREATE TABLE fact_sales (
    sales_id        SERIAL PRIMARY KEY,
    date_key        DATE        REFERENCES dim_date(date_key),
    product_key     INT         REFERENCES dim_product(product_key),
    region_key      INT         REFERENCES dim_region(region_key),
    quantity_sold   INT,
    sales_amount    NUMERIC(12,2)
);
```

1.3 Insert Sample Data

Overview: We will insert:

- 3–4 products across 2 categories,
- 3–4 regions,
- several Q1 2025 dates,
- and around 12 sales rows with varying quantities and amounts.

Listing 3: Sample inserts for dimensions and fact table

```
-- 1.3.1 dim_date rows
INSERT INTO dim_date (date_key, year, quarter, month_num, month_name) VALUES
(’2025-01-05’, 2025, ’Q1’, 1, ’January’),
(’2025-01-12’, 2025, ’Q1’, 1, ’January’),
(’2025-02-03’, 2025, ’Q1’, 2, ’February’),
(’2025-02-18’, 2025, ’Q1’, 2, ’February’),
(’2025-03-07’, 2025, ’Q1’, 3, ’March’),
(’2025-03-21’, 2025, ’Q1’, 3, ’March’);

-- 1.3.2 dim_product rows
INSERT INTO dim_product (product_name, category, subcategory) VALUES
(’Alpha Phone’,     ’Electronics’, ’Mobile’),
(’Beta Phone’,      ’Electronics’, ’Mobile’),
(’Gamma Laptop’,    ’Electronics’, ’Computer’),
(’Delta Headphones’,’Accessories’, ’Audio’);
```

```
-- 1.3.3 dim_region rows
INSERT INTO dim_region (region_name, country, city) VALUES
(’North America’, ’USA’, ’New York’),
(’North America’, ’USA’, ’San Francisco’),
(’Asia Pacific’,  ’Singapore’, ’Singapore’),
(’Europe’,        ’Germany’, ’Berlin’);


-- 1.3.4 fact_sales rows
INSERT INTO fact_sales (date_key, product_key, region_key, quantity_sold,
    sales_amount)
VALUES
(’2025-01-05’, 1, 1, 10, 8000.00),
(’2025-01-05’, 2, 1,  6, 4200.00),
(’2025-01-12’, 1, 2,  4, 3200.00),
(’2025-01-12’, 3, 2,  2, 3000.00),
(’2025-02-03’, 3, 3,  5, 9500.00),
(’2025-02-03’, 4, 3, 15, 2250.00),
(’2025-02-18’, 1, 3,  8, 6400.00),
(’2025-02-18’, 2, 4, 12, 8400.00),
(’2025-03-07’, 3, 4,  3, 5700.00),
(’2025-03-07’, 4, 4, 20, 3000.00),
(’2025-03-21’, 2, 1, 11, 7700.00),
(’2025-03-21’, 4, 2, 25, 3750.00);
```

1.4 Basic Check Query

Listing 4: Validating schema via star join

```
SELECT
    fs.sales_id,
    d.month_name,
    d.quarter,
    p.product_name,
    p.category,
    r.region_name,
    r.country,
    r.city,
    fs.quantity_sold,
    fs.sales_amount
FROM fact_sales fs
JOIN dim_date    d ON fs.date_key    = d.date_key
JOIN dim_product p ON fs.product_key = p.product_key
JOIN dim_region  r ON fs.region_key  = r.region_key
ORDER BY fs.sales_id;
```

Explanation: We denormalize (join fact + dimensions) to obtain a human-readable, analysis-friendly dataset. This "**star join**" forms the foundation for OLAP-style aggregation and reporting. *(Ref: Lec7 — Data Warehouse)*

1.5 Exercises

Q1.1 Write a query that returns **total sales_amount per category** for all data.

Q1.2 Write a query that returns **total quantity_sold per country and city**.

Q1.3 Explain why `fact_sales` stores numeric measures (`quantity_sold`, `sales_amount`) while `dim_product` stores descriptive text (e.g., `category`).
(Answer in 2–3 sentences.)

1.6 Reference Answers

Listing 5: Answer to Q1.1 — Total sales by category

```
-- A1.1
SELECT p.category,
       SUM(fs.sales_amount) AS total_sales_amount
```

```
FROM fact_sales fs
JOIN dim_product p ON fs.product_key = p.product_key
GROUP BY p.category;
```

Listing 6: Answer to Q1.2 — Total quantity by country and city

```
-- A1.2
SELECT r.country,
       r.city,
       SUM(fs.quantity_sold) AS total_quantity
FROM fact_sales fs
JOIN dim_region r ON fs.region_key = r.region_key
GROUP BY r.country, r.city
ORDER BY r.country, r.city;
```

A1.3 (Short Text): Fact tables store numeric, additive measures that we aggregate (e.g., `SUM`, `AVG`). Dimension tables store descriptive attributes that we use for grouping and filtering (e.g., `country`, `product_name`). This separation of measures and dimensions is the foundation of **star schema modeling** in data warehouses and OLAP systems. *(Ref: Lec7 — Data Warehouse)*

Slice and Dice in PostgreSQL

2.1 Concept Recap

Slice: Fix one value of one dimension — for example, analyze only data for `Time = January 2025`. *(Ref: Lec7 — Data Warehouse)*

Dice: Filter on a range of values on one or more dimensions — for example, `Q1 2025 AND Region IN ('North America','Asia Pacific')`. *(Ref: Lec7 — Data Warehouse)*

Summary: Both operations are implemented via `WHERE` filters in SQL but represent core conceptual moves in OLAP — they “cut” the data cube along one or multiple dimensions to isolate specific subspaces for analysis. *(Ref: Lec7 — Data Warehouse)*

Next: We’ll practice both slice and dice operations in PostgreSQL.

2.2 SLICE Examples

Slice Example 1: “Only February 2025”

Listing 7: Slice by Time dimension (February 2025)

```sql
SELECT
    d.date_key,
    d.month_name,
    p.product_name,
    r.region_name,
    fs.quantity_sold,
    fs.sales_amount
FROM fact_sales fs
JOIN dim_date    d ON fs.date_key    = d.date_key
JOIN dim_product p ON fs.product_key = p.product_key
JOIN dim_region  r ON fs.region_key  = r.region_key
WHERE d.year = 2025
  AND d.month_name = ’February’   -- slice on Time dimension
ORDER BY fs.sales_id;
```

Explanation: This query **slices the cube** to a single “flat face” — all rows where `month_name=’February’`. Analytically: *“What happened in February 2025 across all products and regions?”*

—

Slice Example 2: “Only Asia Pacific”

Listing 8: Slice by Region dimension (Asia Pacific)

```sql
SELECT
    r.region_name,
    r.country,
    r.city,
    p.product_name,
    fs.quantity_sold,
    fs.sales_amount
FROM fact_sales fs
JOIN dim_region  r ON fs.region_key  = r.region_key
JOIN dim_product p ON fs.product_key = p.product_key
WHERE r.region_name = 'Asia Pacific';  -- slice by Region dimension
```

Explanation: This query extracts all sales for a single region. Conceptually, it's a slice along the **Region** dimension.

—

2.3 DICE Examples

Concept: The **DICE** operation selects a sub-cube defined by *multiple values or ranges* on one or more dimensions.

Dice Example 1: Time in Q1 + Region subset

Listing 9: Dice on Q1 2025 and Region subset

```sql
-- Business question: Show Q1 2025 sales in North America OR Asia Pacific (ignore
    Europe)
SELECT
    d.quarter,
    d.month_name,
    r.region_name,
    p.category,
    fs.quantity_sold,
    fs.sales_amount
FROM fact_sales fs
JOIN dim_date    d ON fs.date_key    = d.date_key
JOIN dim_product p ON fs.product_key = p.product_key
JOIN dim_region  r ON fs.region_key  = r.region_key
WHERE d.year = 2025
  AND d.quarter = ’Q1’                      -- time range subset
  AND r.region_name IN (’North America’,’Asia Pacific’); -- region subset
```

Explanation: We filter by a *set of values* rather than a single value, forming a smaller multi-dimensional cube. This is the essence of a **DICE** operation.

Dice Example 2: Category + Date Range

Listing 10: Dice on Category and Date range

```
-- Business question: What were Electronics sales in January and February 2025?
SELECT
    d.month_name,
    p.category,
    SUM(fs.sales_amount) AS total_sales
FROM fact_sales fs
JOIN dim_date    d ON fs.date_key    = d.date_key
JOIN dim_product p ON fs.product_key = p.product_key
WHERE p.category = ’Electronics’
  AND d.year = 2025
  AND d.month_num BETWEEN 1 AND 2   -- restrict to JanFeb
GROUP BY d.month_name, p.category
ORDER BY d.month_name;
```

Explanation: This DICE query limits analysis to two months and one category, enabling focused trend comparisons.

2.4 Exercises

Q2.1 (Slice) Write a query that returns **total sales_amount by product_name** only for the **Asia Pacific** region.

Q2.2 (Dice) Write a query that returns **total quantity_sold by city** for **Q1 2025** and **category = ’Electronics’**.

Q2.3 (Explain) In your own words:

- When is **SLICE** more appropriate than **DICE**?
- When is **DICE** more powerful than **SLICE**?

2.5 Reference Answers

Listing 11: A2.1 Slice – Fix region_name = ’Asia Pacific’

```sql
-- A2.1 Slice: Fix region_name = ’Asia Pacific’
SELECT
    p.product_name,
    SUM(fs.sales_amount) AS total_sales_amount
FROM fact_sales fs
JOIN dim_product p ON fs.product_key = p.product_key
JOIN dim_region  r ON fs.region_key  = r.region_key
WHERE r.region_name = ’Asia Pacific’
GROUP BY p.product_name
ORDER BY total_sales_amount DESC;
```

Listing 12: A2.2 Dice – Restrict multiple dimensions at once

```sql
-- A2.2 Dice: restrict multiple dimensions at once
SELECT
    r.city,
    SUM(fs.quantity_sold) AS total_quantity
FROM fact_sales fs
JOIN dim_date    d ON fs.date_key    = d.date_key
JOIN dim_product p ON fs.product_key = p.product_key
JOIN dim_region  r ON fs.region_key  = r.region_key
WHERE d.year = 2025
  AND d.quarter = ’Q1’
  AND p.category = ’Electronics’
GROUP BY r.city
ORDER BY total_quantity DESC;
```

A2.3 Text: **SLICE** is used when zooming into exactly one value on a single dimension (e.g., only February, only Asia Pacific). **DICE** is used when filtering by a defined subset or range (e.g., January–February, multiple regions, or selected categories). DICE is more flexible because it applies multiple filters and retains multiple values within each filter.

Drill-Down and Roll-Up in PostgreSQL

3.1 Concept Recap

Definitions: **Drill-down** moves from summarized to more detailed levels (e.g., Year → Quarter → Month → Day). **Roll-up** is the reverse: aggregating detail back up (Month → Quarter → Year). They represent opposite directions of hierarchy navigation in OLAP.

(Ref: Lec7 — Data Warehouse)

Typical OLAP Cube Hierarchies:

- **Time:** Year → Quarter → Month → Day
- **Geography:** Region → Country → City
- **Product:** Category → Subcategory → Product

Implementation in SQL: In PostgreSQL, we simulate these operations using successive `GROUP BY` levels, or with built-in OLAP extensions such as `ROLLUP`, `CUBE`, and `GROUPING SETS`.

3.2 Drill-Down Step-by-Step

Step 1 — Year-Level Summary

Listing 13: Drill-down Step 1: Year summary

```
SELECT
    d.year,
    SUM(fs.sales_amount) AS total_sales_amount
FROM fact_sales fs
JOIN dim_date d ON fs.date_key = d.date_key
GROUP BY d.year
ORDER BY d.year;
```

Step 2 — Year → Quarter (Drill deeper)

Listing 14: Drill-down Step 2: Year and Quarter summary

```
SELECT
    d.year,
    d.quarter,
    SUM(fs.sales_amount) AS total_sales_amount
FROM fact_sales fs
JOIN dim_date d ON fs.date_key = d.date_key
GROUP BY d.year, d.quarter
ORDER BY d.year, d.quarter;
```

Step 3 — Year → Quarter → Month (Even deeper)

Listing 15: Drill-down Step 3: Year, Quarter, and Month summary

```
SELECT
    d.year,
    d.quarter,
    d.month_name,
    SUM(fs.sales_amount) AS total_sales_amount
FROM fact_sales fs
JOIN dim_date d ON fs.date_key = d.date_key
```

```
GROUP BY d.year, d.quarter, d.month_name
ORDER BY d.year, d.quarter, MIN(d.month_num);
```

Explanation: Each query adds a lower-level dimension column to the `GROUP BY` clause. This mimics what analysts do when "drilling in" to view progressively finer-grained details. *(Ref: Lec7 — Data Warehouse)*

3.3 Roll-Up with GROUPING SETS / ROLLUP

Concept: PostgreSQL supports advanced OLAP aggregation constructs: `GROUPING SETS`, `ROLLUP`, and `CUBE`.

ROLLUP(a, b, c) automatically produces grouped combinations:

$$(a, b, c),\ (a, b),\ (a),\ ()$$

This allows hierarchical summaries — moving from detailed to aggregated levels — in one query.

Listing 16: Roll-Up by Year, Quarter, Month

```
SELECT
    d.year,
    d.quarter,
    d.month_name,
    SUM(fs.sales_amount) AS total_sales_amount
FROM fact_sales fs
JOIN dim_date d ON fs.date_key = d.date_key
GROUP BY ROLLUP (d.year, d.quarter, d.month_name)
ORDER BY d.year, d.quarter, d.month_name;
```

Interpretation:

- Rows where `month_name IS NULL` but `quarter IS NOT NULL` represent **quarter totals**.

- Rows where `quarter IS NULL` but `year IS NOT NULL` represent **year totals**.
- The row where `year IS NULL` represents the **grand total**.

This simulates both **roll-up** (less detail) and **drill-down** (more detail) within a single SQL query.

—

3.4 More Drill Examples

Drill by Geography

Listing 17: Drill by Region → Country → City

```sql
SELECT
    r.region_name,   -- high level
    r.country,
    r.city,          -- detailed level
    SUM(fs.sales_amount) AS total_sales_amount
FROM fact_sales fs
JOIN dim_region r ON fs.region_key = r.region_key
GROUP BY r.region_name, r.country, r.city
ORDER BY r.region_name, r.country, r.city;
```

Drill by Product Hierarchy

Listing 18: Drill by Product Category → Subcategory → Product

```sql
SELECT
    p.category,
    p.subcategory,
    p.product_name,
    SUM(fs.sales_amount) AS total_sales
FROM fact_sales fs
JOIN dim_product p ON fs.product_key = p.product_key
GROUP BY p.category, p.subcategory, p.product_name
ORDER BY p.category, p.subcategory, p.product_name;
```

Explanation: Each level added to the GROUP BY hierarchy enables analysts to explore data from broader summaries down to specific items — a core OLAP exploration pattern.

3.5 Exercises

Q3.1 Drill-Down: Write a query that shows **total quantity_sold by year, quarter, and product_name.**

Q3.2 Roll-Up: Use ROLLUP to compute:

- total sales per (region_name, country, city)
- plus **country subtotals**
- plus **region subtotals**
- plus **grand total**.

Hint: use ROLLUP(r.region_name, r.country, r.city).

Q3.3 Short Answer: Explain in one paragraph how **drill-down** supports business decision-making.

—

3.6 Reference Answers

Listing 19: A3.1 Drill-Down: Year → Quarter → Product

```
-- A3.1 Drill-Down: year -> quarter -> product
SELECT
    d.year,
    d.quarter,
    p.product_name,
    SUM(fs.quantity_sold) AS total_qty
FROM fact_sales fs
JOIN dim_date    d ON fs.date_key    = d.date_key
JOIN dim_product p ON fs.product_key = p.product_key
```

```
GROUP BY d.year, d.quarter, p.product_name
ORDER BY d.year, d.quarter, p.product_name;
```

Listing 20: A3.2 Roll-Up: Region, Country, City

```
-- A3.2 Roll-Up by region/country/city
SELECT
    r.region_name,
    r.country,
    r.city,
    SUM(fs.sales_amount) AS total_sales
FROM fact_sales fs
JOIN dim_region r ON fs.region_key = r.region_key
GROUP BY ROLLUP (r.region_name, r.country, r.city)
ORDER BY r.region_name, r.country, r.city;
```

A3.3 Text: **Drill-down** enables managers to move from summarized performance metrics to detailed, root-cause insights. For instance, if Q1 revenue is under target, analysts can drill from **Year** → **Quarter** → **Month** → **City** → **Product** to pinpoint which markets or items are underperforming. This capability underlies OLAP's value: providing rapid, multidimensional insight for better business decisions.

Pivot (Rotate) / Crosstab in PostgreSQL (HEAVY FOCUS)

4.1 Concept Recap

Concept: A **Pivot (Rotate)** operation in OLAP swaps dimensions between rows and columns to create a new 2D view. *Example:* Swap `Product Region`, so regions become rows and products become columns.

(Ref: Lec7 — Data Warehouse)

Cross-tab layout: Rows represent one dimension, columns represent another, and cells contain aggregated measures such as `SUM(sales_amount)`.

Techniques in PostgreSQL:

1. **Conditional Aggregation:** using `CASE WHEN ... THEN ... END`. Works in any SQL dialect.
2. **crosstab():** from the `tablefunc` extension — the most "official" PostgreSQL pivot.

We will explore both methods through several examples.

4.2 Conditional Aggregation Pivot

Goal: Columns = `product_name`, Rows = `region_name`, Cells = `SUM(sales_amount)`.

Listing 21: Conditional Aggregation Pivot: Region × Product

```
SELECT
    r.region_name,

    SUM(CASE WHEN p.product_name = ’Alpha Phone’      THEN fs.sales_amount ELSE 0
     END) AS alpha_phone_sales,
    SUM(CASE WHEN p.product_name = ’Beta Phone’       THEN fs.sales_amount ELSE 0
     END) AS beta_phone_sales,
    SUM(CASE WHEN p.product_name = ’Gamma Laptop’     THEN fs.sales_amount ELSE 0
     END) AS gamma_laptop_sales,
    SUM(CASE WHEN p.product_name = ’Delta Headphones’ THEN fs.sales_amount ELSE 0
     END) AS delta_headphones_sales,

    SUM(fs.sales_amount) AS region_total
FROM fact_sales fs
JOIN dim_product p ON fs.product_key = p.product_key
JOIN dim_region  r ON fs.region_key  = r.region_key
GROUP BY r.region_name
```

```
ORDER BY r.region_name;
```

Explanation: We “rotated” `product_name` values into columns. Before the pivot: `product_name` was a row attribute. After the pivot: each product becomes its own column — a textbook OLAP rotation.

—

4.3 Conditional Aggregation Pivot by Month

Goal: Columns = `month_name`, Rows = `category`, Cell = `SUM(quantity_sold)`.

Listing 22: Pivot: Category × Month (Conditional Aggregation)

```
SELECT
    p.category,

    SUM(CASE WHEN d.month_name = 'January'  THEN fs.quantity_sold ELSE 0 END) AS
   jan_qty,
    SUM(CASE WHEN d.month_name = 'February' THEN fs.quantity_sold ELSE 0 END) AS
   feb_qty,
    SUM(CASE WHEN d.month_name = 'March'    THEN fs.quantity_sold ELSE 0 END) AS
   mar_qty,

    SUM(fs.quantity_sold) AS q1_total_qty
FROM fact_sales fs
JOIN dim_product p ON fs.product_key = p.product_key
JOIN dim_date    d ON fs.date_key    = d.date_key
WHERE d.year = 2025
GROUP BY p.category
ORDER BY p.category;
```

Note: This produces the familiar “time across columns” layout used in most BI dashboards.

—

4.4 Creating a Pivot with `crosstab()`

Step 0 — Enable Extension

```
CREATE EXTENSION IF NOT EXISTS tablefunc;
```

Step 1 — Prepare Source Query The `crosstab()` function expects:

- `row_name` – row dimension
- `category_name` – will become columns
- `cell_value` – aggregated measure

Listing 23: Pivot Source Query (Region × Product)

```
SELECT
    r.region_name    AS row_name,
    p.product_name   AS column_name,
    SUM(fs.sales_amount) AS cell_value
FROM fact_sales fs
JOIN dim_region  r ON fs.region_key  = r.region_key
JOIN dim_product p ON fs.product_key = p.product_key
GROUP BY r.region_name, p.product_name
ORDER BY r.region_name, p.product_name;
```

paragraphStep 2 — Apply `crosstab()`

Listing 24: Pivot with crosstab(): Region × Product

```
SELECT *
FROM crosstab(
    $$SELECT
          r.region_name    AS row_name,
          p.product_name   AS column_name,
          SUM(fs.sales_amount) AS cell_value
      FROM fact_sales fs
      JOIN dim_region  r ON fs.region_key  = r.region_key
      JOIN dim_product p ON fs.product_key = p.product_key
      GROUP BY r.region_name, p.product_name
      ORDER BY r.region_name, p.product_name$$,

    $$SELECT DISTINCT product_name
      FROM dim_product
      ORDER BY product_name$$
) AS ct(
    region_name TEXT,
    "Alpha Phone" NUMERIC,
    "Beta Phone" NUMERIC,
    "Delta Headphones" NUMERIC,
    "Gamma Laptop" NUMERIC
);
```

Notes:

- The output column order must match the order of the second subquery.
- This produces a fully pivoted result — the canonical “cross-tab” in PostgreSQL.

4.5 Pivot with Time Across Columns Using crosstab()

Listing 25: Pivot Months Across Columns (Category × Month)

```
SELECT *
FROM crosstab(
    $$SELECT
          p.category       AS row_name,
          d.month_name     AS column_name,
          SUM(fs.quantity_sold)::INT AS cell_value
      FROM fact_sales fs
      JOIN dim_product p ON fs.product_key = p.product_key
      JOIN dim_date    d ON fs.date_key    = d.date_key
      WHERE d.year = 2025
      GROUP BY p.category, d.month_name
      ORDER BY p.category, d.month_name$$,

    $$SELECT DISTINCT month_name
      FROM dim_date
      WHERE year = 2025
      ORDER BY month_num$$
) AS ct(
    category TEXT,
    January  INT,
    February INT,
    March    INT
);
```

Explanation: Each row = category; columns = months; cells = total units sold. This is the format most managers prefer in Excel or BI dashboards.

4.6 Extra Pivot: Region Across Columns, Month as Rows

Listing 26: Pivot: Month × Region (Sales Amount)

```
SELECT *
FROM crosstab(
    $$SELECT
          d.month_name       AS row_name,
          r.region_name      AS column_name,
          SUM(fs.sales_amount) AS cell_value
      FROM fact_sales fs
      JOIN dim_date   d ON fs.date_key   = d.date_key
      JOIN dim_region r ON fs.region_key = r.region_key
      WHERE d.year = 2025
      GROUP BY d.month_name, r.region_name, MIN(d.month_num)
      ORDER BY MIN(d.month_num), r.region_name$$,

    $$SELECT DISTINCT region_name
      FROM dim_region
      ORDER BY region_name$$
) AS ct(
    month_name TEXT,
    "Asia Pacific"   NUMERIC,
    "Europe"         NUMERIC,
    "North America"  NUMERIC
);
```

We grouped by `MIN(d.month_num)` to maintain natural month order.

4.7 Exercises

Q4.1 Conditional Aggregation Pivot: Rows = city; Columns = category (“Electronics”, “Accessories”); Cell = SUM(sales_amount). Use `CASE WHEN`, not `crosstab()`.

Q4.2 Crosstab Pivot: Use `crosstab()` to pivot product_name × quarter with SUM(sales_amount). Prepare for future quarters (Q1–Q4).

Q4.3 Short Answer: Why is Pivot also called “Rotate” in OLAP?

4.8 Reference Answers

Listing 27: A4.1 Conditional Aggregation Pivot (City × Category)

```
SELECT
    r.city,
    SUM(CASE WHEN p.category = ’Electronics’ THEN fs.sales_amount ELSE 0 END) AS
    electronics_sales,
    SUM(CASE WHEN p.category = 'Accessories' THEN fs.sales_amount ELSE 0 END) AS
    accessories_sales,
    SUM(fs.sales_amount) AS city_total
FROM fact_sales fs
JOIN dim_product p ON fs.product_key = p.product_key
JOIN dim_region  r ON fs.region_key  = r.region_key
GROUP BY r.city
ORDER BY r.city;
```

Listing 28: A4.2 Pivot with crosstab(): Product × Quarter

```
SELECT *
FROM crosstab(
    $$SELECT
          p.product_name        AS row_name,
          d.quarter             AS column_name,
          SUM(fs.sales_amount)  AS cell_value
      FROM fact_sales fs
      JOIN dim_product p ON fs.product_key = p.product_key
      JOIN dim_date    d ON fs.date_key    = d.date_key
      GROUP BY p.product_name, d.quarter
      ORDER BY p.product_name, d.quarter$$,

    $$SELECT DISTINCT quarter
      FROM dim_date
      ORDER BY quarter$$
) AS ct(
    product_name TEXT,
    "Q1" NUMERIC,
    "Q2" NUMERIC,
    "Q3" NUMERIC,
    "Q4" NUMERIC
);
```

A4.3 Text: **Pivot = Rotate** because we literally rotate dimensions — turning rows into columns (or vice versa) to reveal a new 2D view. It produces Excel-like pivot tables that make large datasets instantly interpretable.

Mini Project + Review Quiz

5.1 Mini Project (Integrated Task)

Goal: Analyze Q1 2025 data comparing **Electronics vs Accessories** by region and month, drill into weak regions, and produce an Excel-style pivot (month × region revenue).

5.2 Step 1 — Dice to Q1 2025

Listing 29: Dice: Filter to Q1 2025

```
CREATE TEMP VIEW q1_2025_sales AS
SELECT
    fs.*,
    d.year,
    d.quarter,
    d.month_num,
    d.month_name,
    p.product_name,
    p.category,
    r.region_name,
    r.country,
    r.city
FROM fact_sales fs
JOIN dim_date    d ON fs.date_key    = d.date_key
JOIN dim_product p ON fs.product_key = p.product_key
JOIN dim_region  r ON fs.region_key  = r.region_key
```

```
WHERE d.year = 2025
  AND d.quarter = 'Q1';
```

Explanation: This applies a DICE operation — selecting only Q1 2025 rows.

5.3 Step 2 — Category × Region Summary

Listing 30: Category × Region Summary

```
SELECT
    category,
    region_name,
    SUM(sales_amount) AS total_sales_amount
FROM q1_2025_sales
GROUP BY category, region_name
ORDER BY category, total_sales_amount DESC;
```

Interpretation: Shows which categories perform best by region.

5.4 Step 3 — Drill-Down (Region → City)

Listing 31: Drill-Down: Asia Pacific to City Level

```
SELECT
    region_name,
    country,
    city,
    category,
    SUM(sales_amount) AS total_sales_amount,
    SUM(quantity_sold) AS total_units
FROM q1_2025_sales
WHERE region_name = 'Asia Pacific'
GROUP BY region_name, country, city, category
```

```
ORDER BY total_sales_amount DESC;
```

Explanation: This is a DRILL-DOWN — moving from region to city-level detail.

5.5 Step 4 — Final Pivot (Month × Region)

Listing 32: Final Crosstab: Month × Region

```
SELECT *
FROM crosstab(
    $$SELECT
          month_name          AS row_name,
          region_name         AS column_name,
          SUM(sales_amount)   AS cell_value
      FROM q1_2025_sales
      GROUP BY month_name, month_num, region_name
      ORDER BY month_num, region_name$$,

    $$SELECT DISTINCT region_name
      FROM q1_2025_sales
      ORDER BY region_name$$
) AS ct(
    month_name TEXT,
    "Asia Pacific"   NUMERIC,
    "Europe"         NUMERIC,
    "North America"  NUMERIC
);
```

Interpretation: This delivers an executive-friendly pivot table — month rows, region columns, and total revenue as values. It's a "rotated" 2D cube view — the essence of OLAP PIVOT.

5.6 Final Review Quiz

Q5.1 Concept Definitions: Define each (1–2 sentences):

- Slice
- Dice
- Drill-down
- Roll-up
- Pivot (Rotate)

Q5.2 SQL Reasoning: Identify the OLAP operation (Slice / Dice / Drill / Pivot):

a) `SELECT * FROM q1_2025_sales WHERE region_name = ’Europe’;`

b) `SELECT year, quarter, month_name, SUM(sales_amount) FROM q1_2025_sales GROUP BY year, quarter, month_name;`

c) `SELECT city, SUM(CASE WHEN category=’Electronics’ THEN sales_amount ELSE 0 END) AS electronics, SUM(CASE WHEN category=’Accessories’ THEN sales_amount ELSE 0 END) AS accessories FROM q1_2025_sales GROUP BY city;`

d) `SELECT * FROM q1_2025_sales WHERE category=’Electronics’ AND region_name IN (’Asia Pacific’,’North America’) AND month_name IN (’January’,’February’);`

Q5.3 Short Essay: Why is Pivot valuable for non-technical managers? Mention spreadsheets and presentations.

5.7 Reference Answers

A5.1 Definitions:

- **Slice:** Filter to one value on one dimension (e.g., February 2025).
- **Dice:** Filter to a sub-cube using multiple dimension conditions (e.g., Q1 2025, Asia Pacific + North America).

- **Drill-down:** Move from summarized to detailed view (Year → Quarter → Month).
- **Roll-up:** Aggregate from detail to higher levels (Month → Quarter → Year).
- **Pivot (Rotate):** Swap dimensions between rows and columns to form a 2D cross-tab report.

A5.2 Classification:

a) Slice

b) Drill-down

c) Pivot

d) Dice

A5.3 Essay: **Pivot** is invaluable for non-technical managers because it turns raw data into visual, spreadsheet-like tables rows, columns, and totals that make patterns obvious. It "rotates" the cube into an Excel-style cross-tab, perfect for dashboards and presentations.

(Ref: Lec7 — Data Warehouse)