Chapter 3

3.1 Multidimensional Model

OLTP (Online Transaction Processing):

- **Characteristics:**
  - Traditional database systems like the ones studied in **Chap. 2** do not satisfy the requirements of data analysis.
  - Designed and tuned to support the daily operations of an organization.
  - Their primary concern is to ensure fast, concurrent access to data.
  - This requires transaction processing and concurrency control capabilities, as well as recovery techniques that guarantee data consistency.
  - These systems are known as **operational databases or Online Transaction Processing (OLTP)** systems.
  - The OLTP paradigm is focused on transactions.
  - Since OLTP systems must support heavy transaction loads, their design should prevent update anomalies.
  - Thus, OLTP databases are highly normalized using the techniques studied in Chap. 2.
  - Thus, they perform poorly when executing complex queries that need to join many relational tables together or to aggregate large volumes of data.
  - Typical OLTP query like the above would require accessing only a few records of the database (and normally will return a few tuples).
  - Typical operational databases contain detailed data and do not include historical data.
- **Advantages:**
  - Ensure fast, concurrent access to data.
  - Guarantee data consistency.
  - Prevent update anomalies.
  - Efficient for simple transactions that access a few records.
- **Disadvantages:**
  - Perform poorly when executing complex queries that need to join many relational tables together or to aggregate large volumes of data.
  - Do not include historical data.

OLAP (Online Analytical Processing)

- **Characteristics:**
  - A paradigm specifically oriented to analyze the data in organizational databases to support decision making.
  - The OLAP paradigm is focused on queries, in particular, analytical queries.
  - OLAP-oriented databases should support a heavy query load.
  - Typical OLAP queries involve aggregation, and thus, processing them will require, most of the time, traversing all the records in a database table.
  - Indexing techniques aimed at OLTP are not efficient in this case: new indexing and query optimization techniques are required for OLAP.

- Normalization is not good for these queries, since it partitions the database into many tables. Reconstructing the data would require a high number of joins.
- Therefore, the need for a different database model to support OLAP was clear and led to the notion of **data warehouses**.

➢ **Advantages:**

- Specifically oriented to analyze the data in organizational databases to support decision making.
- Efficient for heavy analytical query loads.

➢ **Disadvantages:**

- OLTP indexing techniques are not efficient.
- Normalized structures make analytical queries inefficient due to excessive joins.

Aspect	**OLTP (Online Transaction Processing)**	**OLAP (Online Analytical Processing)**
Purpose	Supports **daily operations** of an organization	Supports **data analysis and decision making**
Focus	**Transactions** (Insert, Update, Delete)	**Analytical queries** (Summarize, Aggregate, Drill-down)
Query Type	Simple, short **read/write** queries (e.g., retrieve or update a few records)	Complex **read-only** queries (e.g., aggregations, trends, reports)
Data Volume Accessed	Small number of records per query	Large number of records per query
Normalization	**Highly normalized** (to prevent anomalies)	**Denormalized** or star/snowflake schema (for faster querying)
Data Type	**Current and detailed** operational data	**Historical and summarized** data
Indexing	Uses **traditional indexing** techniques	Needs **specialized indexing** (bitmap, materialized views)
Performance Goal	Fast **concurrent** transactions	Fast **analytical** response times
Concurrency Control	High (supports many users updating data)	Low (mainly read operations)
Example Use Case	ATM transactions, order processing, inventory updates	Sales trend analysis, customer behavior analysis, forecasting
Data Consistency	Strong emphasis on **data consistency and integrity**	May tolerate minor inconsistencies for **faster query performance**
System Type	Operational Database System	Data Warehouse System
Historical Data	**Not stored**	**Stored and analyzed**
Advantages	- Fast concurrent access - Ensures data consistency - Avoids anomalies	- Efficient for complex queries - Supports decision making

Aspect	OLTP (Online Transaction Processing)	OLAP (Online Analytical Processing)
Disadvantages	- Poor at complex queries - No historical data	- Not suitable for frequent updates - Needs complex setup for large-scale data

Data Warehouses

➢ **Description from the paragraph:**

- The need for a different database model to support OLAP was clear and led to the notion of data warehouses.
- **Data warehouses** are (usually) large repositories that consolidate data from different sources (internal and external to the organization).
- Are updated off-line (although as we will see, this is not always the case in modern data warehouse systems).
- Follow the multidimensional data model.
- Being dedicated analysis databases, data warehouses can be designed and optimized to efficiently support OLAP queries.
- In addition, data warehouses are also used to support other kinds of analysis tasks, like reporting, data mining, and statistical analysis.

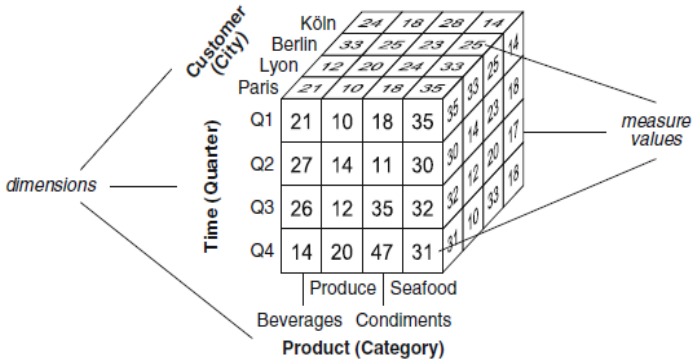


Fig. 3.1 A three-dimensional cube for sales data with dimensions Product, Time, and Customer, and a measure Quantity

➢ **Multidimensional Model and Data Cube**

- Data warehouses and OLAP systems are based on the multidimensional model, which views data in an n-dimensional space, usually called a data cube or a hypercube.
- A data cube is defined by dimensions and facts.

📐 Dimensions

- **Dimensions** are perspectives used to analyze the data.
- The cube has three dimensions**: Product, Time, and Customer.**
- A **dimension level** represents the granularity, or level of detail, at which measures are represented for each dimension of the cube.
- In the example, sales figures are aggregated to the levels Category, Quarter, and City, respectively.
- Instances of a dimension are called **members.**
- For example, Seafood and Beverages are members of the Product dimension at the Category level.
- Dimensions also have associated attributes describing them.
- For example, the Product dimension could contain attributes such as ProductNumber and UnitPrice, which are not shown in the figure.

📊 Facts and Measures

- The cells of a data cube, or facts, have associated numeric values (we will see later that this is not always the case), called measures.
- These measures are used to evaluate quantitatively various aspects of the analysis at hand.
- Each number shown in a cell of the data cube in Fig. 3.1 represents a measure Quantity, indicating the number of units sold (in thousands) by category, quarter, and customer's city.
- A data cube typically contains several measures.
- Another measure, not shown in the figure, could be Amount, indicating the total sales amount.

📉 Sparsity and Density

- A data cube may be sparse or dense depending on whether it has measures associated with each combination of dimension values.
- In the case of Fig. 3.1, this depends on whether all products are bought by all customers during the period of time considered.
- Not all customers may have ordered products of all categories during all quarters of the year.
- In real-world applications, cubes are typically sparse.

3.1.1 Hierarchies

Granularity of a Data Cube

- We have said that the granularity of a data cube is determined by the combination of the levels corresponding to each axis of the cube.
- In **Fig. 3.1,** the dimension levels are indicated between parentheses: Category for the Product dimension, Quarter for the Time dimension, and City for the Customer dimension.

Need for Multiple Levels of Detail

- In order to extract strategic knowledge from a cube, it is necessary to view its data at several levels of detail.
- In our example, an analyst may want to see the sales figures at a finer granularity, such as at the month level, or at a coarser granularity, such as at the customer's country level.

Hierarchies and Dimension Schema

- Hierarchies allow this possibility by defining a sequence of mappings relating lower-level, detailed concepts to higher-level, more general concepts.
- Given two related levels in a hierarchy, the lower level is called the child and the higher level is called the parent.
- The hierarchical structure of a dimension is called the dimension schema, while a dimension instance comprises the members at all levels in a dimension.

Examples of Dimension Hierarchies

- **Figure 3.2** shows the simplified hierarchies for our cube example.
- In the Product dimension, products are grouped in categories.
- For the Time dimension, the lowest granularity is Day, which aggregates into Month, which in turn aggregates into Quarter, Semester, and Year.
- Similarly, for the Customer dimension, the lowest granularity is Customer, which aggregates into City, State, Country, and Continent.

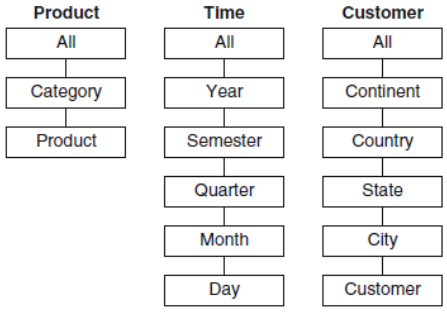


Fig. 3.2 Hierarchies of the Product, Time, and Customer dimensions

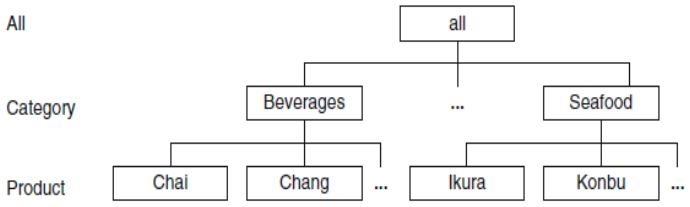


Fig. 3.3 Members of a hierarchy Product → Category

🏔️ Distinguished Level: All

- It is usual to represent the top of the hierarchy with a distinguished level called All corresponding category.
- All categories are grouped under a member called all, which is the only member of the distinguished level All.
- This member is used for obtaining the aggregation of measures for the whole hierarchy, that is, for obtaining the total sales for all products.

Example Hierarchies:

📍 Customer Dimension Hierarchy

From **Customer → City → State → Country → Continent → All**

Level	Example
Customer	*John Smith*
City	*New York City*
State	*New York*
Country	*United States*
Continent	*North America*
All	*All Customers*

Another example:

- Customer: *Ayesha Rahman*
- City: *Dhaka*
- State: *Dhaka Division*
- Country: *Bangladesh*
- Continent: *Asia*
- All: *All Customers*

🕒 Time Dimension Hierarchy

From **Day → Month → Quarter → Semester → Year → All**

Level	Example
Day	*15 March 2025*
Month	*March 2025*
Quarter	*Q1 2025*
Semester	*First Half 2025*
Year	*2025*
All	*All Time*

Another example:

- Day: *10 October 2024*
- Month: *October 2024*
- Quarter: *Q4 2024*
- Semester: *Second Half 2024*

- Year: *2024*
- All: *All Time*

📦 **Product Dimension Hierarchy**

From **Product → Category → All**

Level	Example
Product	*Coca-Cola 500ml*
Category	*Beverages*
All	*All Products*

Another example:

- Product: *Atlantic Salmon Fillet*
- Category: *Seafood*
- All: *All Products*

🧮 **Example OLAP Query Aggregation Scenarios**

1. **Total sales by customer city in Q1(Quarter 1) 2025**
    - Aggregated at: City (Customer) + Quarter (Time)
2. **Total sales for all products in March 2024**
    - Aggregated at: All (Product) + Month (Time)
3. **Quantity sold of Beverages category in Bangladesh in 2023**
    - Aggregated at: Category (Product) + Country (Customer) + Year (Time)
4. **Total sales of all products to all customers in all time**
    - Aggregated at: All (Product) + All (Customer) + All (Time)

3.1.2 Measures

📌 **Aggregation in Data Cubes**

- Each measure in a cube is associated with an aggregation function that combines several measure values into a single one.
- Aggregation of measures takes place when one changes the level of detail at which data in a cube are visualized.
- This is performed by traversing the hierarchies of the dimensions.
- For example, if we use the Customer hierarchy in Fig. 3.2 for changing the granularity of the data cube in Fig. 3.1 from City to Country, then the sales figures for all customers in the same country will be aggregated using, for example, the SUM operation.
- Similarly, total sales figures will result in a cube containing one cell with the total sum of the quantities of all products, that is, this corresponds to visualizing the cube at the All level of all dimension hierarchies.

✅ Summarizability

- Summarizability refers to the correct aggregation of cube measures along dimension hierarchies, in order to obtain consistent aggregation results.

✔️ Conditions for Summarizability:

- **Disjointness of instances**:
  - The grouping of instances in a level with respect to their parent in the next level must result in disjoint subsets.
  - For example, in the hierarchy of Fig. 3.3, a product cannot belong to two categories. If this were the case, each product sales would be counted twice, one for each category.
  - **Violation Example**: A single product belonging to both **Seafood** and **Beverages** categories would **violate disjointness**, causing **double-counting** in aggregation.
  - **Correct Example**: Product Atlantic Salmon belongs **only** to **Seafood**, not to any other category.
- **Completeness**:
  - All instances must be included in the hierarchy and each instance must be related to one parent in the next level.
  - For example, the instances of the Time hierarchy in Fig. 3.2 must contain all days in the period of interest, and each day must be assigned to a month. If this condition were not satisfied, the aggregation of the results would be incorrect, since there would be dates for which sales will not be counted.
  - **Violation Example**: A day (e.g., *March 15, 2025*) is not assigned to any **Month**.
  - **Correct Example**: *March 15, 2025* belongs to **March**, which belongs to **Q1**, then **First Half**, then **2025**, then **All Time**.
- **Correctness**:
  - It refers to the correct use of the aggregation functions.
  - As explained next, measures can be of various types, and this determines the kind of aggregation function that can be applied to them.
  - **Violation Example**: Summing **unit price** or **exchange rate** — leads to wrong results.
  - **Correct Example**:
    a) Use Average for **Unit Price**
    b) Use Max for **Exchange Rate**

📊 Types of Measures (Based on Aggregation Possibility)

1. **Additive measures** can be meaningfully summarized along all the dimensions, using addition.
   - These are the most common type of measures.
   - For example, the measure Quantity in the cube of Fig. 3.1 is additive: it can be summarized when the hierarchies in the Product, Time, and Customer dimensions are traversed.
   - **Example Measure**: Quantity Sold

- **Applicable Aggregation**: SUM
- Add **Quantity Sold** across:
  - Product hierarchy: Category → All
  - Time hierarchy: Day → Month → Quarter → Semester → Year → All
  - Customer hierarchy: City → State → Country → Continent → All
- **Scenario**:
  a) Total quantity of Beverages sold in March 2024 to customers in Dhaka.
  b) Total quantity of all products sold globally in 2025.

2. **Semi-additive measures** can be meaningfully summarized using addition along some, but not all, dimensions.
   - A typical example is that of inventory quantities, which cannot be meaningfully aggregated in the Time dimension, for instance, by adding the inventory quantities for two different quarters.
   - **Example Measure**: Inventory Quantity
   - **Applicable Aggregation**:
     - SUM along Product and Customer dimensions
     - But **not** along Time (use Average, Last Value, etc.)
   - **Example Using Fig. *3.2***:
     - Can **sum inventory by product** or **by customer country**, but not across time periods.
   - **Scenario**:
     a) Average inventory of Seafood in Q1 2025.
     b) Sum of inventory per category (e.g., Beverages) per city, as of March 2025.
3. **Nonadditive measures** cannot be meaningfully summarized using addition across any dimension.
   - Typical examples are item price, cost per unit, and exchange rate.
   - **Example Measures**: Unit Price, Exchange Rate
   - **Applicable Aggregation**: Average, Min, Max, or Most Recent Value
   - **Example Using Fig. *3.2***:
     a) Average **price** of products per category
     b) Max **exchange rate** per year
   - **Scenario**:
     a) Average unit price of products sold in Continent = Asia during 2024.
     b) Minimum price of products in Category = Seafood during Q1.

⚙️ Choosing Aggregation Functions for Measures

- In order to define a measure, it is necessary to determine the aggregation functions that will be used in the various dimensions.
- This is particularly important in the case of semi-additive and nonadditive measures.
- For example, a semi-additive measure representing inventory quantities can be aggregated computing the average along the Time dimension and computing the sum along other dimensions.
- Averaging can also be used for aggregating nonadditive measures such as item price or exchange rate.

- However, depending on the semantics of the application, other functions such as the minimum, maximum, or count could be used instead.

🧠 Optimization Techniques: Aggregate Precomputation

- In order to allow users to interactively explore the cube data at different granularities, optimization techniques based on aggregate precomputation are used.
- To avoid computing the whole aggregation from scratch each time the data warehouse is queried, OLAP tools implement incremental aggregation mechanisms.
- However, incremental aggregation computation is not always possible, since this depends on the kind of aggregation function used.

📚 Classification of Measures Based on Aggregation Function Type

1. **Distributive measures** are defined by an aggregation function that can be computed in a distributed way.
   - Suppose that the data are partitioned into n sets and that the aggregate function is applied to each set, giving n aggregated values.
   - The function is distributive if the result of applying it to the whole data set is the same as the result of applying a function (not necessarily the same) to the n aggregated values.
   - The usual aggregation functions such as the count, sum, minimum, and maximum are distributive.
   - However, the distinct count function is not.
   - For instance, if we partition the set of measure values {3, 3, 4, 5, 8, 4, 7, 3, 8} into the subsets {3, 3, 4}, {5, 8, 4}, and {7, 3, 8}, summing up the result of the distinct count function applied to each subset gives us a result of 8, while the answer over the original set is 5.
   - **Examples**: **`SUM, COUNT, MIN, MAX`**
   - **Example Using Fig. *3.2***:
     - Total Quantity Sold = SUM of sales per City → per State → per Country → All
     - Maximum Quantity Sold in a City during a Quarter
2. **Algebraic measures** are defined by an aggregation function that can be expressed as a scalar function of distributive ones.
   - A typical example of an algebraic aggregation function is the average, which can be computed by dividing the sum by the count, the latter two functions being distributive.
   - **Examples**: Average = SUM / COUNT
   - **Example**:
     - Average number of items sold per month in 2024 for each category.
3. **Holistic measures** are measures that cannot be computed from other sub-aggregates.
   - Typical examples include **the median, the mode, and the rank**.
   - Holistic measures are expensive to compute, especially when data are modified, since they must be computed from scratch.

- **Examples**: Median, Mode, Rank
- **Example**:
  - *Median quantity sold per product in 2025.*
  - *Most common product category in Q1 2025.*

Type	Measure Example	Aggregation Function	Valid Dimensions	Example Query
Additive	Quantity Sold	SUM	All	Total quantity of Beverages sold in Asia in Q1 2024
Semi-additive	Inventory Level	SUM (some), AVERAGE	Not Time	Average inventory of Seafood in Jan 2025 across all cities
Non-additive	Unit Price	AVERAGE / MAX	None with SUM	Average unit price of products sold in 2025 in the USA
Distributive Func	Quantity, Count	SUM, COUNT	All	Total number of orders per country in 2023
Algebraic Func	Average Quantity	AVG = SUM / COUNT	All	Average quantity sold per product per month
Holistic Func	Median Quantity	MEDIAN, MODE	Hard to compute across dims	Median quantity of orders for Seafood products in Q1

3.2 OLAP Operations

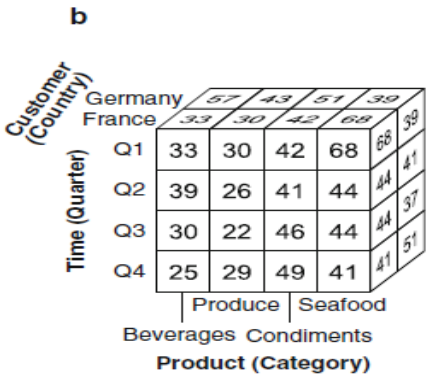
Table 3.1 Summary of the OLAP operations

Operation	Purpose
Add measure	Adds new measures to a cube computed from other measures or dimensions.
Aggregation operations	Aggregate the cells of a cube, possibly after performing a grouping of cells.
Dice	Keeps the cells of a cube that satisfy a Boolean condition over dimension levels, attributes, and measures.
Difference	Removes the cells of a cube that are in another cube. Both cubes must have the same schema.
Drill-across	Merges two cubes that have the same schema and instances using a join condition.
Drill-down	Disaggregates measures along a hierarchy to obtain data at a finer granularity. It is the opposite of the roll-up operation.
Drill-through	Shows data in the operational systems from which the cube was derived.
Drop measure	Removes measures from a cube.
Pivot	Rotates the axes of a cube to provide an alternative presentation of its data.
Recursive roll-up	Performs an iteration of roll-ups over a recursive hierarchy until the top level is reached.
Rename	Renames one or several schema elements of a cube.
Roll-up	Aggregates measures along a hierarchy to obtain data at a coarser granularity. It is the opposite of the drill-down operation.
Roll-up*	Shorthand notation for a sequence of roll-up operations.
Slice	Removes a dimension from a cube by fixing a single value in a level of the dimension.
Sort	Orders the members of a dimension according to an expression.
Union	Combines the cells of two cubes that have the same schema but disjoint members.

Original Cube:

ROLL-UP Operation

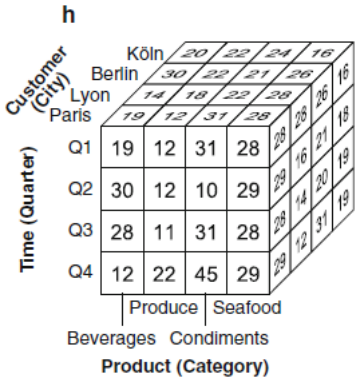
- The roll-up operation aggregates measures along a dimension hierarchy (using an aggregate function) to obtain measures at a coarser granularity.
- **Syntax**:
  `ROLLUP(CubeName, (Dimension → Level)*, AggFunction(Measure)*)`
- In the example given in **Fig. 3.4b**, we applied the operation:
- **(b) Roll-up to the Country level:**
  `ROLLUP(Sales, Customer → Country, SUM(Quantity))`



✅ **Figure 3.4h** shows the result of a **ROLLUP operation**.

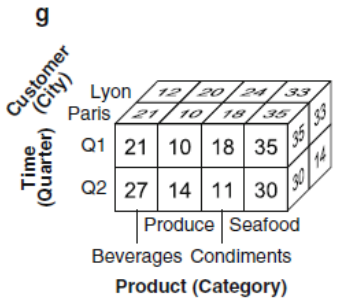
🔍 **Context from your book:**

"Our user now wants to compare the sales quantities in 2012 with those in 2011. For this, she needs the cube in Fig. 3.4h, which has the same structure as the one for 2012 given in Fig. 3.4a."



This means:

- She **aggregated** the 2011 sales data cube (**like in Fig. 3.4g**)
- `DICE(Sales, (Customer.City = 'Paris' OR Customer.City = 'Lyon') AND (Time.Quarter = 'Q1' OR Time.Quarter = 'Q2'))` **fig 3.4(g)**



- Into a higher-level view **by Customer.Country and Time.Quarter**,
- To **match the structure** of **Fig. 3.47a** (which is the 2012 cube at the same level).

✅ **Figure 3.4h Operation:**

```
ROLLUP(Sales2011, Customer → Country, Time → Quarter, SUM(Quantity))
```

🔁 **ROLLUP* Operation**

- The ROLLUP* operation provides a shorthand notation for a sequence of ROLLUP operations.
- **Syntax**:
  ```
  ROLLUP*(CubeName, [(Dimension → Level)*], AggFunction(Measure)*)
  ```
- *For example, the total quantity by quarter can be obtained as follows:*
  ```
  ROLLUP*(Sales, Time → Quarter, SUM(Quantity))
  ```
- *If dimensions are not specified:*
  ```
  ROLLUP*(Sales, SUM(Quantity))
  ```

Counting in ROLLUP

- A usual need when applying a roll-up operation is to count the number of members in one of the dimensions removed from the cube.
- *Example query:*

```
ROLLUP*(Sales, Time → Quarter, COUNT(Product) AS ProdCount)
```

RECROLLUP Operation (Recursive Hierarchy)

- Used to aggregate measures over recursive hierarchies by iteratively performing roll-ups until the top level is reached.
- **Syntax**:

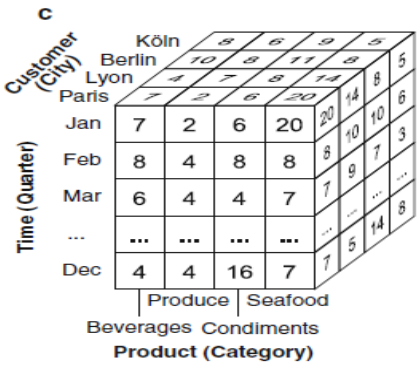
```
RECROLLUP(CubeName, Dimension → Level, Hierarchy, AggFct(Measure)*)
```

DRILLDOWN Operation

- Performs the inverse of the roll-up operation.
- **Syntax**:

```
DRILLDOWN(CubeName, (Dimension → Level)*)
```

- In our example given in **Fig. 3.4c**, we applied the operation:



Drill Down to the Month level:

```
DRILLDOWN(Sales, Time → Month)
```

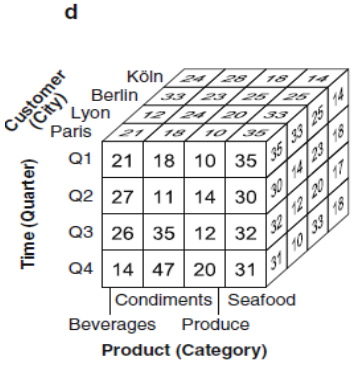
SORT Operation

- Returns a cube where the members of a dimension have been sorted.
- **Syntax**:

```
SORT(CubeName, Dimension, (Expression [ {ASC | DESC | BASC | BDESC} ])*)
```

- *Example:*

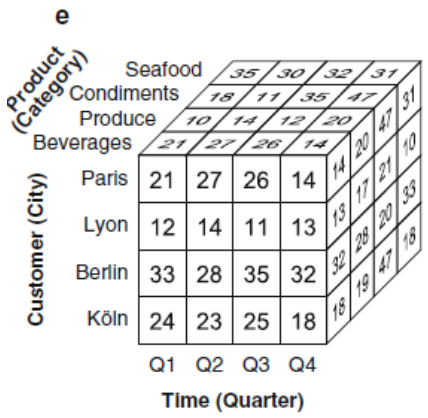
```
SORT(Sales, Product, ProductName)
SORT(SalesByQuarter, Time, Quantity DESC)
```



PIVOT (Rotate) Operation

- Rotates the axes of a cube.
- **Syntax**:
  `PIVOT(CubeName, (Dimension → Axis)*)`
- *Example:*
  `PIVOT(Sales, Time → X, Customer → Y, Product → Z)`

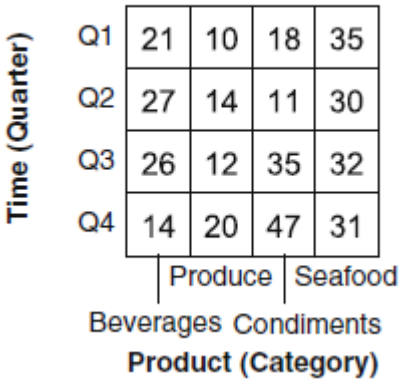
e



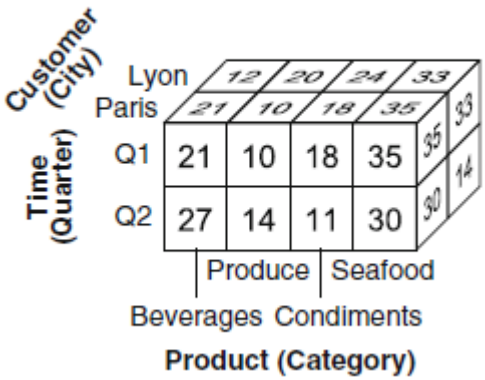
SLICE Operation

- Removes a dimension in a cube.
- **Syntax**:
  `SLICE(CubeName, Dimension, Level = Value)` -> fig 3.4(f)

f



g



- *Example:*
  `SLICE(Sales, Customer, City = 'Paris')`

DICE Operation

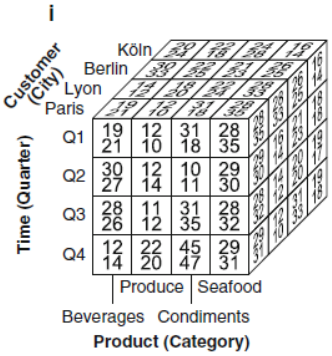
- Keeps the cells in a cube that satisfy a Boolean condition.
- **Syntax**:
  `DICE(CubeName, Φ)`
- *Example:*
  `DICE(Sales, (Customer.City = 'Paris' OR Customer.City = 'Lyon') AND (Time.Quarter = 'Q1' OR Time.Quarter = 'Q2'))` -> fig 3.4(g)

📝 RENAME Operation

- Returns a cube where some schema elements or members have been renamed.
- **Syntax**:
  `RENAME(CubeName, ({SchemaElement | Member} → NewName)*)`
- *Examples:*
  `RENAME(Sales, Sales → Sales2012, Quantity → Quantity2012)`
  `RENAME(Sales, Customer.all → AllCustomers)`

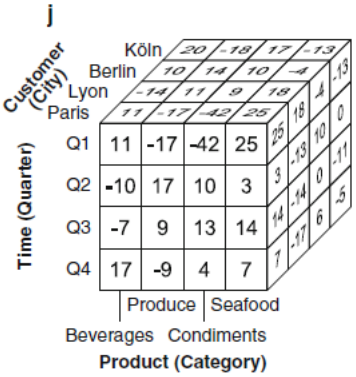
🔁 DRILLACROSS Operation

- Combines cells from two data cubes that have the same schema and instances.
- **Syntax**:
  `DRILLACROSS(CubeName1, CubeName2, [Condition])`
- *Example:*
  `Sales2011-2012 ← DRILLACROSS(Sales2011, Sales2012)`



➕ ADDMEASURE Operation

- Adds new measures to the cube computed from other measures or dimensions.
- **Syntax**:
  `ADDMEASURE(CubeName, (NewMeasure = Expression, [AggFct])*)`
- *Examples:*
  - `ADDMEASURE(Sales2011-2012, PercentageChange = (Quantity2011-Quantity2012)/Quantity2011)`
  - `ADDMEASURE(Sales, MovAvg = AVG(Quantity) OVER Time 2 CELLS PRECEDING)`
  - `ADDMEASURE(Sales, YTDQuantity = SUM(Quantity) OVER Time ALL CELLS PRECEDING)`



DROPMEASURE Operation

- Removes one or several measures from a cube.
- **Syntax**:
  `DROPMEASURE(CubeName, Measure*)`
- *Example:*
  `DROPMEASURE(Sales2011-2012, Quantity2011, Quantity2012)`

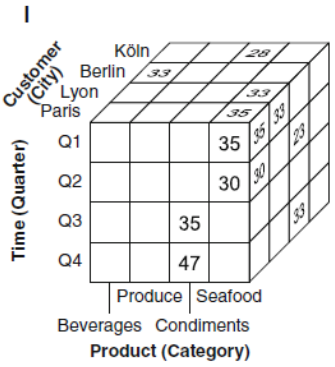
Aggregation Without Roll-Up

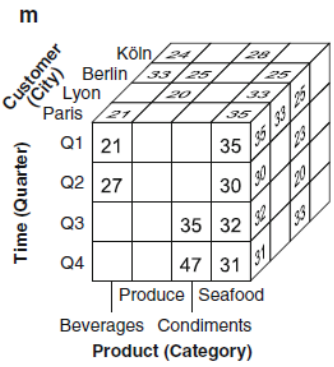
- **Syntax**:
  `AggFunction(CubeName, Measure) [BY Dimension*]`
- *Examples:*
  - `SUM(Sales, Quantity)` → single cell
  - `SUM(Sales, Quantity) BY Time` → 1D cube
  - Total Sales by Quarter and City: `SUM(Sales, Quantity) BY Time, Customer` → 2D cube (Fig. 3.4k)
  - `MAX(Sales, Quantity)` → cell with max value and coordinates (Q4, Condiments, Paris)
  - Maximum sales by quarter and city: `MAX(Sales, Quantity) BY Time, Customer` → Fig. 3.4l
  - Top two sales by quarter and city: `MAX(Sales, Quantity, 2) BY Time, Customer` → Fig. 3.4m

k

Time (Quarter)	Paris	Berlin	Lyon	Köln
Q1	84	89	106	84
Q2	82	77	93	79
Q3	105	72	65	88
Q4	112	61	96	102

Customer (City)





Standard Aggregation Functions

➤ **Syntax:**
`AggFunction(CubeName, Measure) [BY Dimension*]`

➤ **Common Functions:**

- **SUM, AVG, COUNT, MIN, MAX**
- **Extended versions**:
  - `MAX(Cube, Measure, n)` – returns top-n
  - `TOPPERCENT(Cube, Measure, percent)` – cumulative percentage
  - `RANK, DENSERANK`

🔢 Classification of Aggregation Functions

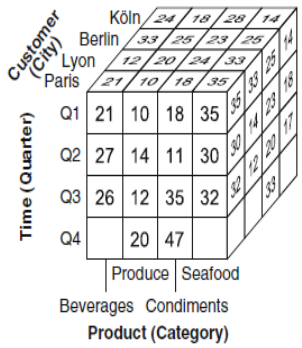
➤ **Cumulative Aggregation Functions**: Compute the measure value of a cell from several other cells.
Examples: SUM, COUNT, AVG

➤ **Filtering Aggregation Functions**: Filter the members of a dimension that appears in the result.
Examples: MIN, MAX

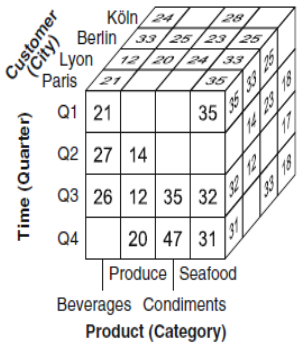
🔝 Top Percent & Rank Aggregations

- **TOPPERCENT (ordered by Quarter)**:
  `TOPPERCENT(Sales, Quantity, 70) BY City, Category ORDER BY Quarter ASC` → Fig. 3.4n
- **TOPPERCENT (ordered by Quantity)**:
  `TOPPERCENT(Sales, Quantity, 70) BY City, Category ORDER BY Quantity DESC` → Fig. 3.4o
- **RANK**:
  `RANK(Sales, Time) BY Category, City ORDER BY Quantity DESC` → Fig. 3.4p

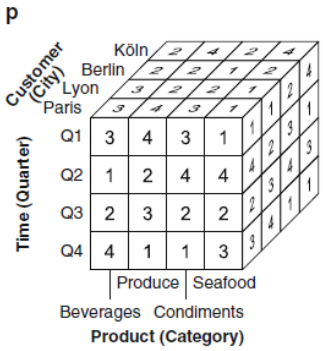
n



o



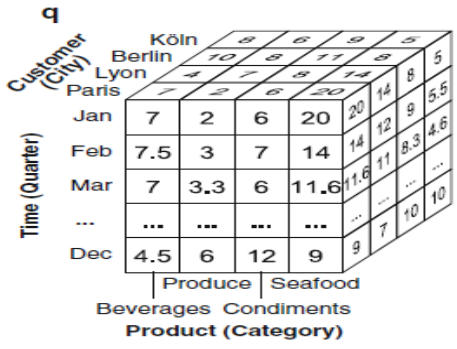
p



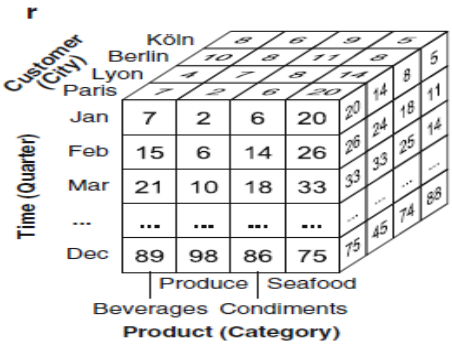
📈 Window Functions (Moving Average, YTD)

- **3-Month Moving Average**:
  `ADDMEASURE(Sales, MovAvg = AVG(Quantity) OVER Time 2 CELLS PRECEDING)` → Fig. 3.4q
- **Year-to-Date Sum**:
  `ADDMEASURE(Sales, YTDQuantity = SUM(Quantity) OVER Time ALL CELLS PRECEDING)` → Fig. 3.4r

q



r

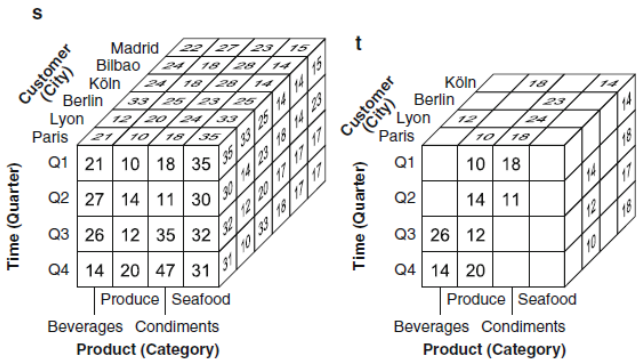


🔗 UNION Operation

- Merges two cubes that have the same schema but disjoint instances.
- **Syntax**:
  `UNION(Cube1, Cube2)`
- *Example:*
  `UNION(Sales, SalesSpain)` → **Fig. 3.4s**
  `UNION(Sales, SalesCountry)` → displays both city and country levels

➖ DIFFERENCE Operation

- Removes the cells in a cube that exist in another one.
- **Syntax**:
  `DIFFERENCE(Cube1, Cube2)`
- *Example:*
  `DIFFERENCE(Sales, TopTwoSales)` → **Fig. 3.4t**



🔎 DRILL-THROUGH Operation

- Allows one to move from data at the bottom level in a cube to data in the operational systems from which the cube was derived.

3.3 Data Warehouses

📦 Definition of a Data Warehouse

- A data warehouse is a repository of integrated data obtained from several sources for the specific purpose of multidimensional data analysis.
- More technically, a data warehouse is defined as a collection of:
  - **Subject-oriented**
  - **Integrated**
  - **Nonvolatile**
  - **Time-varying**

  data to support management decisions.

🔑 Characteristics of a Data Warehouse

1. Subject Oriented

- Subject oriented means that data warehouses focus on the analytical needs of different areas of an organization.
- These areas vary depending on the kind of activities performed by the organization.
- For example, in the case of a retail company, the analysis may focus on product sales or inventory management.
- In operational databases, on the contrary, the focus is on specific functions that applications must perform, for example, registering sales of products or inventory replenishment.

2. Integrated

- Integrated means that data obtained from several operational and external systems must be joined together, which implies solving problems due to differences in data definition and content, such as:
  - Differences in data format and data codification
  - Synonyms (fields with different names but the same data)
  - Homonyms (fields with the same name but different meanings)
  - Multiplicity of occurrences of data
  - And many others
- In operational databases these problems are typically solved in the design phase.

3. Nonvolatile

- Nonvolatile means that durability of data is ensured by disallowing data modification and removal, thus expanding the scope of the data to a longer period of time than operational systems usually offer.
- A data warehouse gathers data encompassing several years, typically 5–10 years or beyond.
- While data in operational databases is often kept for only a short period of time, for example, from 2 to 6 months, as required for daily operations, and it may be overwritten when necessary.
- **Example:**
  - In an **operational system**:
    - A customer updates their address — the old address is **replaced**.
  - In a **data warehouse (nonvolatile)**:
    - The old address is **kept**, and the new one is **added** with a timestamp.
    - This way, you can see **how the data evolved** over time.

4. Time Varying

- Time varying indicates the possibility of retaining different values for the same information, as well as the time when changes to these values occurred.
- For example, a data warehouse in a bank might store information about the average monthly balance of clients' accounts for a period covering several years.
- In contrast, an operational database may not have explicit temporal support, since sometimes it is not necessary for day-to-day operations and it is also difficult to implement.

🏢 Data Warehouses vs. Data Marts

- A data warehouse is aimed at analyzing the data of an entire organization.
- It is often the case that particular departments or divisions of an organization only require a portion of the organizational data warehouse specialized for their needs.
- For example, a sales department may only need sales data, while a human resources department may need demographic data and data about the employees.
- These departmental data warehouses are called **data marts**.
- However, these data marts are not necessarily private to a department; they may be shared with other interested parts of the organization.

🧱 Bottom-Up vs. Top-Down Approaches

- A data warehouse can be seen as a collection of data marts.
  → (Bottom-up approach)
  - This view represents a bottom-up approach in which a data warehouse is built by first building the smaller data marts and then merging these to obtain the data warehouse.
  - This can be a good approach for organizations not willing to take the risk of building a large data warehouse, which may take a long time to complete, or organizations that need fast results.
- In the classic data warehouse view, data marts are obtained from the data warehouse in a top-down fashion.
  - In this approach, a data mart is sometimes just a logical view of a data warehouse.

Table 3.2 Comparison between operational databases and data warehouses

	Aspect	Operational databases	Data warehouses
1	User type	Operators, office employees	Managers, executives
2	Usage	Predictable, repetitive	Ad hoc, nonstructured
3	Data content	Current, detailed data	Historical, summarized data
4	Data organization	According to operational needs	According to analysis needs
5	Data structures	Optimized for small transactions	Optimized for complex queries
6	Access frequency	High	From medium to low
7	Access type	Read, insert, update, delete	Read, append only
8	Number of records per access	Few	Many
9	Response time	Short	Can be long
10	Concurrency level	High	Low
11	Lock utilization	Needed	Not needed
12	Update frequency	High	None
13	Data redundancy	Low (normalized tables)	High (denormalized tables)
14	Data modeling	UML, ER model	Multidimensional model

🔄 OLTP vs OLAP Systems (Based on Table 3.2)

👨‍💼 Users

- Typically, the users of OLTP systems are operators and employees who perform predefined operations through transactional applications, like payroll systems or ticket reservation systems.
- Data warehouse users, on the other hand, are usually located higher in the organizational hierarchy and use interactive OLAP tools to perform data analysis, for example, to detect salary inconsistencies or most frequently chosen tourist destinations (lines 1–2).

🗂️ Data Type

- Data for OLTP systems should be current and detailed, while data analytics require historical, summarized data (line 3).

🧩 Data Organization

- The way data is organized depends on how OLTP and OLAP systems are used.
- The difference in data organization between OLTP and OLAP systems is due to their different purposes. **OLTP** organizes data based on **operational needs** (for fast, real-time transactions), while **OLAP** organizes data based on **analysis needs** (for efficient querying and reporting).

⚙️ Data Structure and Access Pattern

- From a more technical viewpoint, data structures for OLTP are optimized for rather small and simple transactions, which will be carried out frequently and repeatedly.
- In addition, data access for OLTP requires reading and writing data files.
- For example, in the Northwind database application, a user may be able to frequently insert new orders, modify old ones, and delete orders if customers cancel them.
- Thus, the number of records accessed by an OLTP transaction is usually small (e.g., the records involved in a particular sales order).

- On the other hand, data structures for OLAP must support complex aggregation queries, thus requiring access to all the records in one or more tables, which will translate in long, complex SQL queries.
- Furthermore, OLAP systems are not so frequently accessed as OLTP systems.
- For example, a system handling purchase orders is frequently accessed, while performing analysis of orders may not be that frequent.
- Also, data warehouse records are usually accessed in read mode (lines 5–8).

⏱️ **Query Time**

- OLTP systems usually have a short query response time, provided the appropriate indexing structures are defined, while complex OLAP queries can take a longer time to complete (line 9).

👥 **Concurrency**

- OLTP systems have normally a high number of concurrent accesses and therefore require locking or other concurrency management mechanisms to ensure safe transaction processing (lines 10–11).
- On the other hand, OLAP systems are read only, and therefore queries can be submitted and computed concurrently, with no locking or complex transaction processing requirements.
- Further, the number of concurrent users in an OLAP system is usually low.

🔁 **Data Refresh / Update**

- OLTP systems are constantly being updated online through transactional applications, while OLAP systems are updated off-line periodically.

🧱 **Modeling Differences**

- This leads to different modeling choices.
- OLTP systems are modeled using UML or some variation of the ER model studied in Chap. 2.
    - Such models lead to a highly normalized schema, adequate for databases that support frequent transactions, to guarantee consistency and reduce redundancy.
- OLAP designers use the multidimensional model, which, at the logical level (as we will see in Chap.5), leads in general to a denormalized database schema, with a high level of redundancy, which favors query processing (lines 12–14).

3.4 Data Warehouse Architecture

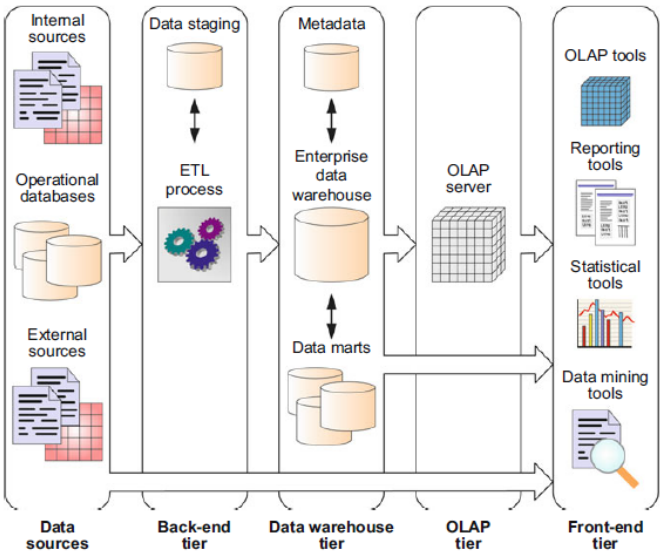


Fig. 3.5 Typical data warehouse architecture

The architecture (Fig. 3.5) consists of four tiers:

- **Back-End Tier**
- **Data Warehouse Tier**
- **OLAP Tier**
- **Front-End Tier**

3.4.1 Back-End Tier

- The back-end tier is composed of extraction, transformation, and loading (ETL) tools
- Used to feed data into the data warehouse from operational databases and other data sources, which can be internal or external from the organization,
- and a data staging area, which is an intermediate database where all the data integration and transformation processes are run before the loading of the data into the data warehouse.

ETL Process:

1. **Extraction**
   - Gathers data from multiple, heterogeneous data sources.
   - These sources may be operational databases but may also be files in various formats; they may be internal to the organization or external to it.
   - In order to solve interoperability problems, data are extracted whenever possible using application programming interfaces (APIs) such as ODBC and JDBC.
2. **Transformation**
   - Modifies the data from the format of the data sources to the warehouse format.
   - Includes:
     - Cleaning (removes errors and inconsistencies)
     - Integration (reconciles data from different sources)
     - Aggregation (summarizes data based on warehouse granularity)
3. **Loading**
   - Feeds the data warehouse with the transformed data.

- Includes refreshing the warehouse at specified frequencies — from monthly to near real time.

Data Staging Area:

- ETL processes usually require a data staging area... usually called operational data store.

3.4.2 Data Warehouse Tier

- The data warehouse tier in Fig. 3.5 depicts an enterprise data warehouse and several data marts.
- An enterprise data warehouse is centralized and encompasses an entire organization.
- A data mart is a specialized data warehouse targeted toward a particular functional or departmental area.
- Data in a data mart can be derived from an enterprise data warehouse or collected directly from data sources.

Metadata Repository:

- Metadata can be defined as "data about data."
- Classified into:
  - Business metadata — describes the meaning of data, rules, policies, constraints
  - Technical metadata — describes how data are structured/stored and manipulated

Contents of metadata repository:

- Metadata describing the structure of the data warehouse and the data marts.
- Metadata describing the data sources.
- Metadata describing the ETL process.

3.4.3 OLAP Tier

- The OLAP tier in the architecture... is composed of an OLAP server, which presents business users with multidimensional data from data warehouses or data marts.

OLAP Tools and Languages:

- Most database products provide OLAP extensions and related tools...
- Languages:
  - **XMLA**: for exchanging multidimensional data
  - **MDX**: de facto standard query language for OLAP databases
  - **SQL/OLAP**: extension of SQL for analytics

3.4.4 Front-End Tier

- The front-end tier in Fig. 3.5 contains client tools...

Types of Client Tools:

- **OLAP Tools**: Interactive exploration, ad hoc queries
- **Reporting Tools**: Predefined, regular queries; paper-based or interactive
- **Statistical Tools**: Analyze and visualize cube data statistically
- **Data Mining Tools**: Discover patterns, trends, make predictions

3.4.5 Variations of the Architecture

- Some components may be missing in real environments:
  - Enterprise DW without data marts
  - Data marts without enterprise DW (harder to integrate later)
  - No OLAP server — clients access DW directly
  - No DW or OLAP server — virtual DW defines views over operational DBs
  - No staging area if data is high-quality and closely matches warehouse

---

3.5 Data Warehouse Design

- Two major methods:
  - **Top-Down**: Design full DW schema first, derive data marts
  - **Bottom-Up**: Design each data mart, then merge into DW

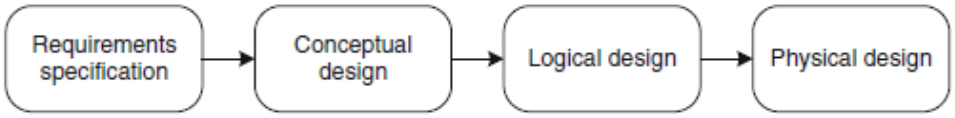


Fig. 3.6 Phases in data warehouse design

➕ **Design Phases (Fig. 3.6):**

- Requirements Specification
- Conceptual Design
- Logical Design
- Physical Design
- Most books use a bottom-up, practitioner's approach (star/snowflake schemas).
- This book uses a **model-based approach**, same as in Chap. 2.

Approaches to Requirements:

- **Analysis-Driven**: User-provided analysis needs
- **Source-Driven**: Analyze data sources; confirm with users
- **Combined (Top-Down/Bottom-Up Analysis)**: Match user needs with source capabilities

---

3.6 Business Intelligence Tools

Commercial & Open Source Tools:

- Major vendors: **Microsoft, Oracle, IBM, Teradata**
- Other popular tools: **SAP, MicroStrategy, Targit**
- Open source: **Pentaho**

3.6.1 Microsoft SQL Server Tools

Components:

- **Analysis Services**:
  - OLAP + data mining capabilities
  - MDX for querying
  - DMX for data mining
- **Integration Services**:
  - ETL — extract, clean, populate DW
- **Reporting Services**:
  - Define, store, deliver reports from DW or cubes

Development Tools:

- SQL Server Data Tools (SSDT): VS-integrated platform
- SQL Server Management Studio (SSMS): Administer all components

Semantic Model:

- **BISM (Business Intelligence Semantic Model)**
  - Two modes:
    - **Multidimensional**: Advanced, large-scale BI apps; MDX
    - **Tabular**: Simple, fast; DAX; supports xVelocity

3.6.2 Pentaho Business Analytics

Two Editions:

- Enterprise (commercial)
- Community (open source)

Main Components:

- **Business Analytics Platform**: End-to-end integration + visualization
- **Analysis Services (Mondrian)**:
  - Relational OLAP, supports MDX
- **Data Mining (Weka)**:
  - Regression, classification, clustering, etc.
- **Data Integration (Kettle)**:
  - GUI-based ETL for local/cloud/cluster
- **Report Designer**:
  - Visual report writer using XML

Additional Design Tools:

- **Schema Workbench**: Design OLAP cubes
- **Aggregation Designer**: Precalculate answers to speed up queries
- **Metadata Editor**: Abstract layer over data sources