UNIT IV XML AND DATAWAREHOUSE

SYLLABUS:

XML Database: XML – XML Schema – XML DOM and SAX Parsers – XSL – XSLT – XPath and XQuery – Data Warehouse: Introduction – Multidimensional Data Modelling – Star and Snow flake Schema – Architecture – OLAP Operations and Queries.

SNAPSHOT:

- XML Database
    - Introduction to XML
    - XML Schema
    - XML Document Object Model (DOM) and Simple API for XML (SAX) Parsers
    - XML Transformation using XSL and XSLT
    - XPath: Navigation and Querying in XML
    - XQuery: Query Language for XML
- Data Warehousing
    - Introduction to Data Warehousing
    - Multidimensional Data Modelling
    - Star Schema and Snowflake Schema
    - Data Warehouse Architecture
    - Online Analytical Processing (OLAP)
    - OLAP Operations: Roll-Up, Drill-Down, Slice and Dice, Pivot
    - OLAP Queries: MDX (Multidimensional Expressions)

CONTENTS:

XML DATABASE

XML INTRODUCTION

- HTML is widely used for formatting and structuring Web documents; it is not suitable for specifying structured data that is extracted from Databases.
- A new language—namely, XML (Extensible Markup Language)—has Emerged as the standard for structuring and exchanging data over the Web.
- XML Can be used to provide information about the structure and meaning of the data in The Web pages. (structuring)

XML HIERARCHICAL (TREE) DATA MODEL

- The basic object in XML is the XML document.
- Two main structuring concepts are used to construct an XML document:
  - elements and attributes.

XML ELEMENT

- Elements are identified in a document by their start tag and end tag.
- The tag names Are enclosed between angled brackets < … >, and end tags are further identified by a Slash, </ … >.

  ```
  <Name>ProductX</Name>
  ```
- A major difference between XML and HTML
  Is that XML tag names are defined to describe the meaning of the data elements in the document, rather than to describe how the text is to be displayed.
- This makes it Possible to process the data elements in the XML document automatically by computer programs.
- Also, the XML tag (element) names can be defined in another document, known as the schema document, to give a semantic meaning to the tag names that can be exchanged among multiple users.

XML ATTRIBUTE

- XML attributes are generally used to describe properties and characteristics of the Elements (tags) within which they appear.
- It is also possible to use XML attributes to
  Hold the values of simple data elements; however, this is generally not recommended.
- An exception to this rule is in cases that need to reference another element in another part of the XML document.
- To do this, it is common to use attribute values in one element as the references.
- This resembles the concept of foreign keys in

Relational databases, and is a way to get around the strict hierarchical model that the
XML tree model implies.

- Complex elements are constructed from other elements hierarchically, whereas Simple elements contain data values.
- In the tree Representation, internal nodes represent complex elements, whereas leaf nodes represent simple elements.
- That is why the XML model is called a tree model or a Hierarchical model.

It is possible to characterize three main types of XML documents:

- **Data-centric XML documents**. These documents have many small data Items that follow a specific structure and hence may be extracted from a Structured database.
- **Document-centric XML documents**. These are documents with large Amounts of text, such as news articles or books. There are few or no structured data elements in these documents.
- **Hybrid XML documents**. These documents may have parts that contain Structured data and other parts that are predominantly textual or unstructured.

XML SCHEMA

- The XML schema language is a standard for specifying the structure of XML documents. It uses the same syntax rules as regular XML documents.
- As with XML DTD, XML schema is based on the tree data model, with elements and Attributes as the main structuring concepts. However, it borrows additional concepts from database and object models, such as keys, references, and identifiers.

1. **Schema descriptions and XML namespaces.**

- It is necessary to identify the Specific set of XML schema language elements (tags) being used by specifying a file stored at a Web site location.
  http://www.w3.org/2001/XMLSchema.
- Each such Definition is called an XML namespace, because it defines the set of commands (names) that can be used.
  <xsd:schema xmlns:xsd=http://www.w3.org/2001/XMLSchema>
  <xsd:element name="company">

2. **Annotations, documentation, and language used.**

- The attribute xml:lang of the Xsd:documentation element specifies the language being used, where 'en' stands For the English language.

```
<xsd:annotation>
<xsd:documentation xml:lang="en">Company Schema (Element
Approach) – Prepared by Babak
Hojabri</xsd:documentation>
</xsd:annotation>
```

3. **Elements and types.**

- In XML schema, the name attribute of the **xsd:element tag** specifies the element name,
- This is further specified to be a Sequence of departments, employees, and projects using the Structure of XML schema.

```
<xsd:sequence>
<xsd:element name="department" type="Department" minOccurs="0"
maxOccurs= "unbounded" />
<xsd:element name="employee" type="Employee" minOccurs="0"
maxOccurs= "unbounded">
<xsd:unique name="dependentNameUnique">
<xsd:selector xpath="employeeDependent" />
<xsd:field xpath="dependentName" />
</xsd:unique>
</xsd:element>
<xsd:element name="project" type="Project" minOccurs="0"
maxOccurs="unbounded" />
</xsd:sequence>
```

4. **First-level elements in the COMPANY database**

- These Elements are named employee, department, and project, and each is specified in an xsd:element tag.

5. **Specifying element type and minimum and maximum occurrences.**

- If we specify a type attribute in an xsd:element, the structure of the element must be described Separately, typically using the xsd:complexType element of XML schema.

```
<xsd:complexType>
```

- The minOccurs and maxOccurs tags are used for specifying lower and upper bounds on the number of occurrences of an element in any XML document that conforms to the schema specifications.

```
<xsd:element name="department" type="Department" minOccurs="0" maxOccurs=
"unbounded" />
```

6. **Specifying keys**

- XML schema, it is possible to specify constraints that Correspond to unique and primary key constraints in a relational database as well as foreign keys.
- The xsd:unique tag specifies elements that Correspond to unique attributes in a relational database.

```
</xsd:unique>
<xsd:unique name="projectNameUnique">
<xsd:selector xpath="project" />
<xsd:field xpath="projectName" />
</xsd:unique>
```

- For specifying primary keys, the tag xsd:key is used instead of Xsd:unique.

```
<xsd:key name="projectNumberKey">
<xsd:selector xpath="project" />
<xsd:field xpath="projectNumber" />
</xsd:key>
```

- For specifying foreign keys, the tag Xsd:keyref.
  When specifying a foreign key, the attribute refer of the xsd:keyref tag specifies the referenced primary key, whereas the tags xsd:selector and xsd:field specify the referencing element type and foreign key.

```
</xsd:keyref>
<xsd:keyref name="employeeSupervisorSSNKeyRef" refer="employeeSSNKey">
<xsd:selector xpath="employee" />
<xsd:field xpath="employeeSupervisorSSN" />
</xsd:keyref>
```

7. **Specifying the structures of complex elements via complex types.**

- The next Part of our example specifies the structures of the complex elements Department, Employee, Project, and Dependent, using the tag xsd:complexType.

```
</xsd:complexType>
<xsd:complexType name="Dependent">
<xsd:sequence>
<xsd:element name="dependentName" type="xsd:string" />
<xsd:element name="dependentSex" type="xsd:string" />
<xsd:element name="dependentBirthDate" type="xsd:date" />
```

<xsd:element name="dependentRelationship" type="xsd:string" />

</xsd:sequence>

</xsd:complexType>

- we specify a type attribute in an xsd:element, the structure of the element must be described Separately, typically using the xsd:complexType element of XML schema.

8. Composite (compound) attributes.

- These could have been Directly embedded within their parent elements.

XML DOM AND SAX PARSERS

XML DOM

- The XML DOM (Document Object Model) is a programming interface for working with XML documents. It provides a tree-like representation of the XML structure, allowing developers to access and manipulate the elements, attributes, and content of an XML document using a set of standard methods and properties.

Here is a detailed explanation of XML DOM and its key concepts:

XML DOCUMENTS:

- XML (eXtensible Markup Language) is a markup language that is widely used for structuring and storing data.
- XML documents consist of a hierarchical structure of elements, which are enclosed in tags and may contain attributes and text content.
- The XML DOM allows you to work with these elements in a programmatic way.

NODE OBJECTS:

- In XML DOM, the XML document is represented as a collection of nodes.
- A node can be an element, attribute, text, comment, or other types of XML nodes.
- Each node is an object with properties and methods that allow you to interact with it.
- The most commonly used node types are:
    - Element nodes: Represent the tags in the XML document.
    - Attribute nodes: Represent the attributes of an element.
    - Text nodes: Contain the text content within an element.
    - Comment nodes: Represent the comments within the XML document.

DOCUMENT OBJECT:

- The Document object represents the entire XML document.
- It provides methods to create, load, and save XML documents, as well as to navigate and modify the nodes within the document.

ELEMENT OBJECT:

- The Element object represents an element in an XML document.
- It provides methods to access and modify the element's attributes and child nodes.

ATTRIBUTE OBJECT:

- The Attribute object represents an attribute of an XML element. It provides methods to access and modify the attribute's value.

TEXT NODE OBJECT:

- The Text object represents the text content within an XML element.
- It provides methods to access and modify the text value.

TRAVERSING THE XML STRUCTURE:

- The XML DOM allows you to traverse the XML structure using methods such as getElementsByTagName(), childNodes, and parentNode.
- These methods enable you to navigate through the nodes, access their properties, and perform operations based on their relationships.

MODIFYING XML DOCUMENTS:

- The XML DOM provides methods to create, modify, and delete nodes in an XML document.
- You can add new elements, attributes, and text nodes, update existing nodes, or remove nodes from the document structure.

PARSING AND SERIALIZATION:

- To work with XML using the DOM, you need to parse an XML document into a DOM structure.
- This can be done using a programming language's XML parser or XML libraries.
- Once the XML document is parsed, you can traverse and manipulate its nodes using the DOM methods.
- After making the desired changes, you can serialize the modified DOM back into an XML document for storage or transmission.

SAX PARSERS

- The SAX (Simple API for XML) parser is a widely used event-based XML parser.
- It is designed to parse large XML documents efficiently by processing the XML data as a stream rather than loading the entire document into memory.

SAX PARSING PROCESS WORKS:

- **Event-Driven Model:** The SAX parser operates on the event-driven model, where it reads the XML document sequentially and triggers events based on the content encountered.
- **Event Handlers**: As the SAX parser reads the XML document, it invokes specific event handlers when certain events occur.
- **Content Handling:** When the SAX parser encounters specific XML elements, it triggers corresponding events, such as startElement, endElement, and characters events.
- **startElement**: This event is triggered when the parser encounters the opening tag of an XML element. It provides information about the element's name, attributes, and namespaces.
- **endElement**: This event is triggered when the parser encounters the closing tag of an XML element.
- **Characters**: This event is triggered when the parser encounters the content between the opening and closing tags of an element. It provides the text data within the element.
- **Event Processing**: The user implements code within the event handlers to process the XML content. For example, when the startElement event occurs, the user-defined code can extract information from the element's attributes or perform specific actions based on the element's name.
- **Memory Efficiency**: One of the key advantages of SAX parsing is its memory efficiency. Since the SAX parser reads the XML document as a stream, it does not require loading the entire document into memory, making it suitable for parsing large XML files.
- **Limitations**: The SAX parser has some limitations. For instance, it does not provide a comprehensive view of the entire XML document at once, as it processes the data sequentially. Additionally, it does not support querying or modifying the XML document structure.

XSL AND XSLT

- XSL (eXtensible Stylesheet Language) is a language used to transform and style XML (eXtensible Markup Language) documents.
- The process typically involves
  - applying XSLT templates to the XML document using XPath expressions to select nodes, and

  - generating XSL-FO output that defines the visual presentation.
  - The XSL-FO output can then be further processed by an XSL-FO processor to generate the final formatted output, such as a PDF or printed document.

It consists of three major components:

- **XSLT (XSL Transformations)**,
  - XSLT is the core component of XSL and is used to transform XML documents into other formats, such as HTML, XHTML, or even another XML format.
  - It enables the separation of content and presentation, allowing you to define rules and templates for how the XML data should be transformed and displayed.
  - XSLT uses XPath to navigate through the XML document and select specific elements or attributes for processing.
- **XPath (XML Path Language),**
  - XPath is a language used to navigate and query XML documents.
  - It provides a syntax for selecting nodes and values from an XML document based on their location or properties.
  - XPath expressions can be used to traverse the XML document's structure, select elements, filter nodes based on conditions, and extract data values.
  - For example, an XPath expression could select all <book> elements that have a price greater than $20 or retrieve the text content of a specific <title> element.
- **XSL-FO (XSL Formatting Objects)**.
  - XSL-FO is a component of XSL that defines a vocabulary for describing the visual formatting of XML documents.
  - It provides a way to specify how the transformed XML should be rendered as a final formatted output, such as a PDF document or printed page.
  - By using XSL-FO, you can control various aspects of the output, such as page size, margins, fonts, colours, and positioning of elements.

XPATH AND XQUERY

- There have been several proposals for XML query languages, and two query language Standards have emerged.
- The first is XPath, which provides language constructs for Specifying path expressions to identify certain nodes (elements) or attributes within An XML document that match specific patterns.
- The second is XQuery, which is a More general query language.

XPATH: SPECIFYING PATH EXPRESSIONS IN XML

- An XPath expression generally returns a sequence of items that satisfy a certain pattern as specified by the expression.
- These items are either values or elements or attributes.
- The names in the XPath expression are node names in the XML document tree that are either tag (element) names or attribute names, possibly with Additional qualifier conditions to further restrict the nodes that satisfy the pattern.
- Two main separators are used when specifying a path: single slash (/) and double Slash (//).
    - A single slash before a tag specifies that the tag must appear as a direct Child of the previous (parent) tag,
    - whereas a double slash specifies that the tag can Appear as a descendant of the previous tag at any level.

EXAMPLE:

For example, if the COMPANY XML document is stored at the location www.company.com/info.XML

1. **/company**
    - the first XPath expression in can be written as
    Doc(www.company.com/info.XML)/company
2. **/company/department**
    - The second example in returns all department nodes (elements) and Their descendant subtrees.
3. **//employee [employeeSalary gt 70000]/employeeName**
    - The third XPath expression illustrates the use of //, which is convenient to use if we do not know the full path name we are searching for, but do know The name of some tags of interest within the XML document.
    - The expression returns all employeeName nodes that are direct children of an employee node, such that the employee node has another child element employeeSalary
    Whose value is greater than 70000.
    - This illustrates the use of qualifier conditions,
    Which restrict the nodes selected by the XPath expression to those that satisfy the condition.
4. **/company/employee [employeeSalary gt 70000]/employeeName**
    - The fourth XPath expression in should return the same result as the previous one, except that we specified the full path name in this example.

5. **/company/project/projectWorker [hours ge 20.0]**

    - The fifth Expression in Figure 12.6 returns all project Worker nodes and their descendant nodes That are children under a path /company/project and have a child node hour with a Value greater than 20.0 hours.
6. It Is also possible to use the wildcard symbol *, which stands for any element, as in the following
   Example, which retrieves all elements that are child elements of the root, regardless of their element type. When wildcards are used, the result can be a sequence of different types of items.
   /company/*
- It is possible to move in multiple Directions from the current node in the path expression. These are known as the Axes of an XPath expression.

    - Child Of the current node (/),
    - descendent or self at any level of the current node (//), and
    - Attribute of the current node (@)
    - Parent,
    - ancestor (at any level),
    - Previous sibling (any node at same level to the left in the tree), and
    - next sibling (any Node at the same level to the right in the tree).

XQUERY: SPECIFYING QUERIES IN XML

- XPath allows us to write expressions that select items from a tree-structured XML Document.
- XQuery has very powerful constructs to specify complex queries.
- Hence, in some ways, it qualifies as a full-fledged programming language.
- The typical form of a query in XQuery is known as a FLWRExpression, which stands for the four main clauses of XQuery and has the following Form:
    - FOR <variable bindings to individual nodes (elements)>
    - LET <variable bindings to collections of nodes (elements)>
        - There can be zero or more instances of the FOR clause, as well as of the LET clause in A single XQuery.
    - WHERE <qualifier conditions>
        - The WHERE clause is optional, but can appear at most once,
    - RETURN <query result specification>
        - the RETURN clause must appear exactly once.

Examples

```
LET $d := doc(www.company.com/info.xml)
FOR $x IN $d/company/project[projectNumber = 5]/projectWorker,
$y IN $d/company/employee
WHERE $x/hours gt 20.0 AND $y.ssn = $x.ssn
RETURN <res> $y/employeeName/firstName, $y/employeeName/lastName,
$x/hours </res>
```

Explanation

- Variables are prefixed with the $ sign. In the above example, $d, $x, and $y Are variables.
- The LET clause assigns a variable to a particular expression for the rest of the Query. In this example, $d is assigned to the document file name.
- The FOR clause assigns a variable to range over each of the individual items in a sequence. In our example, the sequences are specified by path expressions. Hence, $x ranges over projectWorker elements, whereas $y ranges over employee elements.
- The WHERE clause specifies additional conditions on the selection of items. In this example, the first condition selects only those projectWorker elements That satisfy the condition (hours gt 20.0). The second condition specifies a Join condition that combines an employee with a projectWorker only if they Have the same ssn value.
- Finally, the RETURN clause specifies which elements or attributes should be Retrieved from the items that satisfy the query conditions. In this example, it will return a sequence of elements each containing **<firstName, lastName,Hours>** for employees who work more that 20 hours per week on project Number 5.

```
FOR $x IN

Doc(www.company.com/info.xml)

//employee [employeeSalary gt 70000]/employeeName

RETURN <res> $x/firstName, $x/lastName </res>
```

- The second query retrieves the first and last names of employees who earn More than $70,000. The variable $x is bound to each employeeName element that is a Child of an employee element, but only for employee elements that satisfy the qualifier that their employeeSalary value is greater than $70,000. The result retrieves the firstName and lastName child elements of the selected employeeName elements.
- Notice that this is an alternative way of specifying the same query in our earlier example, but without the LET clause.

DATA WAREHOUSE

INTRODUCTION

- A database as a collection of related data and a database system as a database and database software together.
- A data warehouse is also a collection of information as well as a supporting system.
- Traditional databases are transactional (relational, object-oriented, network, Or hierarchical).
- Data warehouses have the distinguishing characteristic that they are Mainly intended for decision-support applications.
- Data Warehouses provide access to data for complex analysis, knowledge discovery, and Decision making.
- Several types of applications—OLAP, DSS, and data mining Applications—are supported.
  - OLAP (online analytical processing) is a term used to describe the analysis of complex data from the data warehouse.
  - DSS (decision-support systems), also known as EIS—executive information systems; support an organization's leading decision makers with higher-level data for complex and important Decisions.
  - Data mining is used for knowledge Discovery
- Traditional relational databases cannot be Optimized for OLAP, DSS, or data mining.

CHARACTERISTICS OF DATA WAREHOUSES

- A data warehouse is frequently a store of integrated data from Multiple sources, processed for storage in a multidimensional model.
- Data warehouses typically support time-series and trend Analysis, both of which require more historical data than is generally maintained in Transactional databases.
- Data warehouses are non-volatile. This Means that information in the data warehouse changes far less often.
- Data warehouse information is refreshed according to a Careful choice of refresh policy, usually incremental.

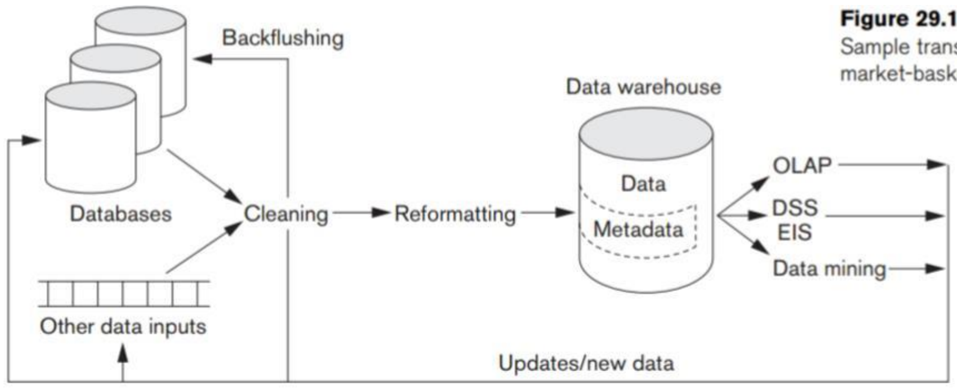


Figure 29.1
Sample transactions in market-basket model.

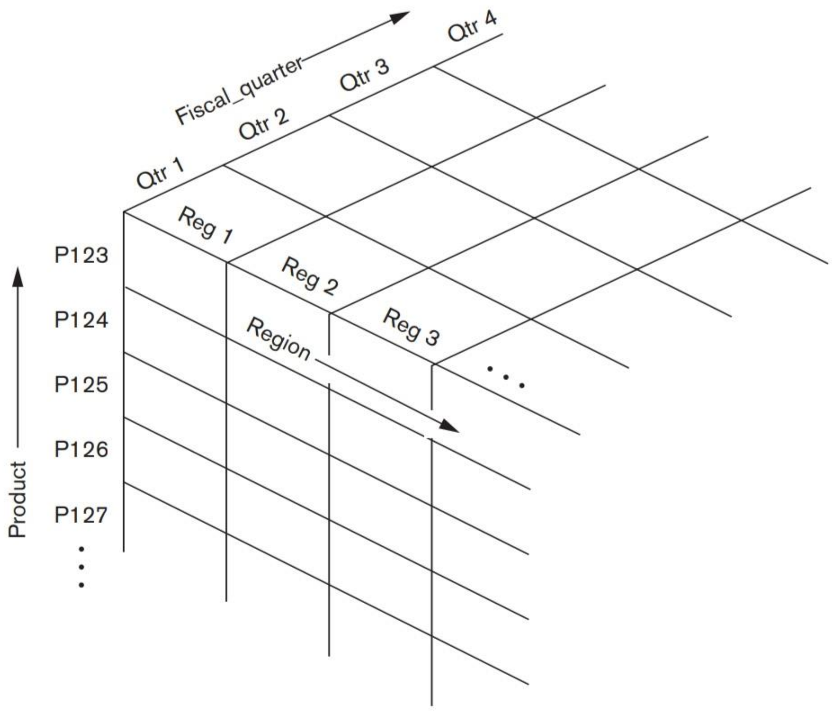
- It shows the entire data warehousing Process, which includes possible cleaning and reformatting of data before loading it into the warehouse.
- This process is handled by tools known as ETL (extraction, Transformation, and loading) tools.

Data warehouses have the following distinctive characteristics

- Multidimensional conceptual view
- Unrestricted cross-dimensional operations
- Client-server architecture
- Multiuser support
- Accessibility
- Transparency
- Intuitive data manipulation
- Flexible reporting

MULTIDIMENSIONAL DATA MODELLING

- Multidimensional models take advantage of inherent relationships in data to populate data in multidimensional matrices called data cubes.



Figu
A thr
data

- One example would be a Spreadsheet of regional sales by product for a particular time period. Products could Be shown as rows, with sales revenues for each region comprising the columns.
- Adding a time dimension, such as an organization's fiscal quarters, would produce a three-dimensional
  Matrix, which could be represented using a data cube.
- Each cell could contain data for a specific product, specific fiscal quarter, and specific region.
- By including additional dimensions, a data Hypercube could be produced.
- The data can be queried directly in any combination of dimensions, bypassing complex database queries.

Tools exist for viewing Data according to the user's choice of dimensions.

Pivoting

- Changing from one-dimensional hierarchy (orientation) to another is easily accomplished in a data cube with a technique called pivoting (also called rotation).
- In this Technique the data cube can be thought of as rotating to show a different orientation of the axes.

- For example, you might pivot the data cube to show regional sales Revenues as rows, the fiscal quarter revenue totals as columns, and the company's Products in the third dimension.

Multidimensional models lend themselves readily to hierarchical views in what is Known as roll-up display and drill-down display.

Roll-up display

- A roll-up display moves up the Hierarchy, grouping into larger units along a dimension.
- Roll-up display that moves from individual products to a coarser-grain of product categories.
- For example, summing Weekly data by quarter or by year.

Drill-down display

- A drill-down display provides the opposite capability, furnishing a finer Grained view,
- perhaps disaggregating country sales by region and then regional sales by subregion and also breaking up products by styles.

The multidimensional storage model involves two types of tables

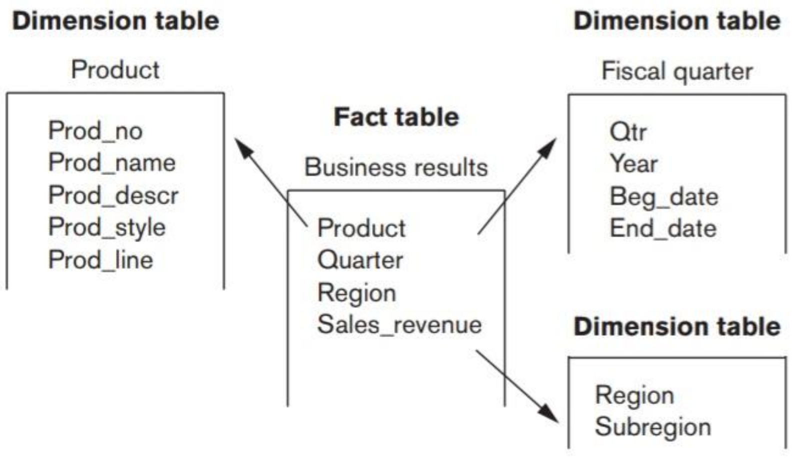
- A **dimension table** consists of tuples of attributes of the dimension.
- A **fact table** can be thought of as having tuples, one per a recorded fact.

STAR AND SNOWFLAKE SCHEMA

Two common multidimensional schemas are the star schema and the snowflake Schema

STAR SCHEMA

- The star schema is a popular data modelling technique used in data warehousing.
- It consists of a central fact table connected to multiple dimension tables, forming a shape resembling a star.
- The fact table contains the primary measures or metrics of interest, while the dimension tables provide additional descriptive attributes for analysing and filtering the data.



KEY CHARACTERISTICS OF THE STAR SCHEMA

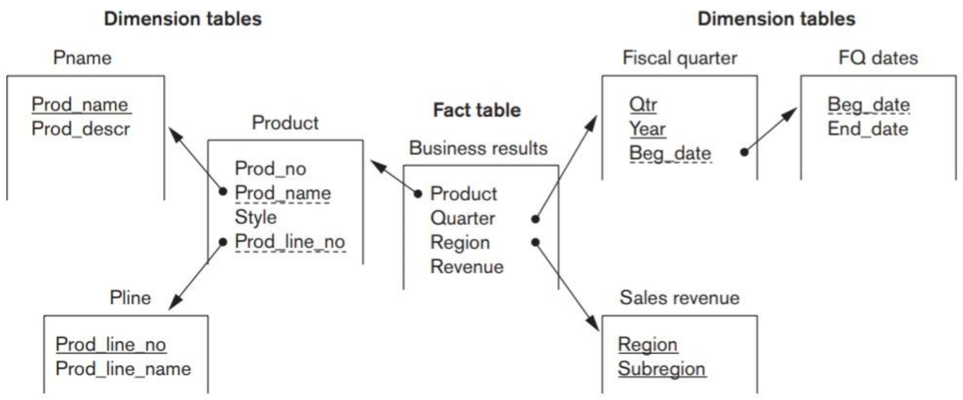
- **Fact Table:**
    - The fact table is the centrepiece of the schema.
    - It contains the quantitative measures or metrics of the business process or event being analysed.
- **Dimension Tables:**
    - Dimension tables provide the descriptive attributes associated with the measures in the fact table.
    - Each dimension table represents a specific aspect of the data, such as time, product, location, or customer.
    - Dimension tables have primary keys, and their primary key values are used as foreign keys in the fact table.
- **Star-like Structure:**
    - The star schema gets its name from its visual representation, where the fact table sits at the centre, and the dimension tables radiate out from it like the points of a star.

BENEFITS OF USING A STAR SCHEMA

- Simplicity:
    - Star schema is straightforward to understand and implement.
    - Its simple structure makes it easy to query and analyse data.
- Performance:
    - Star schema's denormalized design improves query performance by reducing the number of joins required.
    - The fact table holds the primary measures, allowing for efficient aggregation and analysis.
- Flexibility:

- The star schema provides flexibility for ad-hoc querying and reporting, as users can easily navigate across dimensions and aggregate data based on different attributes.

SNOWFLAKE SCHEMA



- The snowflake schema is an extension of the star schema that further normalizes dimension tables.
- It derives its name from the resemblance of the schema's structure to a snowflake, with dimension tables branching out into multiple levels of related tables.
- The snowflake schema reduces data redundancy by breaking down dimension tables into smaller tables.

Key characteristics of the snowflake schema:

- **Normalized Dimension Tables:**
  - Unlike the star schema, where dimension tables are denormalized, the snowflake schema normalizes dimension tables by breaking them down into multiple related tables.
  - This normalization eliminates data redundancy, as shared attributes are stored in separate tables, thereby reducing storage requirements.

- **Hierarchical Structure:**
  - The snowflake schema extends the star schema by introducing additional levels of tables for each dimension.
  - For example, a dimension table in the star schema might be further split into sub-dimension tables in the snowflake schema.

- These sub-dimension tables are connected through primary-foreign key relationships.

Benefits of using a snowflake schema:

a. **Improved Data Integrity**:
    - Normalizing dimension tables in the snowflake schema reduces data redundancy and improves data integrity by eliminating update anomalies.

b. **Storage Efficiency:**
    - By eliminating redundant data, the snowflake schema reduces storage requirements.

c. **Scalability**:
    - The snowflake schema's normalized structure allows for easier scalability.
    - New tables or levels can be added without impacting the existing structure, enabling the schema to accommodate evolving data requirements.

d. **Better Conformance to Database Normalization:**
    - The snowflake schema adheres more closely to the principles of database normalization, which can be beneficial for certain use cases and when integrating with other systems.

S.NO	**Star Schema**	**Snowflake Schema**
1.	In star schema, the fact tables and the dimension tables are contained.	While in snowflake schema, the fact tables, dimension tables as well as sub dimension tables are contained.
2.	Star schema is a top-down model.	While it is a bottom-up model.
3.	Star schema uses more space.	While it uses less space.
4.	It takes less time for the execution of queries.	While it takes more time than star schema for the execution of queries.
5.	In star schema, Normalization is not used.	While in this, both normalization and denormalization are used.
6.	Its design is very simple.	While its design is complex.
7.	The query complexity of star schema is low.	While the query complexity of snowflake schema is higher than star schema.
8.	Its understanding is very simple.	While it is understanding is difficult.
9.	It has a smaller number of foreign keys.	While it has a greater number of foreign keys.
10.	It has high data redundancy.	While it has low data redundancy.

DATA WARE HOUSE ARCHITECTURE

- A data warehouse is a central repository of integrated data that is used for reporting, analysis, and decision-making within an organization.
- It is designed to support the efficient querying and analysis of large volumes of data from multiple sources.

The architecture of a data warehouse typically consists of the following components:

1) **Data Sources:**
   a) Data warehouses integrate data from various sources such as operational databases, transactional systems, external data feeds, spreadsheets, and other data repositories.
   b) These sources can be both internal and external to the organization.
2) **Extraction, Transformation, and Loading (ETL):**
   a) The ETL process involves extracting data from the source systems, transforming it into a consistent and standardized format, and loading it into the data warehouse.
   b) ETL tools are commonly used to automate and streamline this process.
   c) Data cleansing, aggregation, filtering, and validation are performed during the transformation stage to ensure data quality and consistency.
3) **Staging Area:**
   a) The staging area is an intermediate storage area where data from different sources is loaded before it is transformed and loaded into the data warehouse.
   b) It serves as a temporary workspace for data integration and allows for data validation and error handling before loading it into the warehouse.
4) **Data Warehouse Database:**
   a) The data warehouse database is the core component of the architecture.
   b) It is a central repository that stores the integrated, cleansed, and transformed data in a structured format optimized for querying and analysis.
   c) The data warehouse database is typically designed using a star schema or a snowflake schema.
5) **Metadata Repository:**
   a) Metadata is data about the data warehouse, including information about data sources, data transformations, data definitions, and relationships between different data elements.
   b) The metadata repository stores and manages this information, providing a centralized and organized view of the data warehouse's structure and content.

    c) It helps users understand and navigate the data warehouse, improving data governance and facilitating data integration.

6) **Access Tools and Interfaces:**
    a) Data warehouses provide various tools and interfaces for users to access and analyse the data.
    b) These can include reporting tools, query tools, online analytical processing (OLAP) tools, data mining tools, and business intelligence (BI) platforms.
    c) These tools allow users to retrieve and manipulate data, generate reports, perform ad-hoc queries, and gain insights from the data warehouse.
7) **Data Mart:**
    a) A data mart is a subset of a data warehouse that is focused on a specific business function, department, or user group.
    b) Data marts are designed to meet the specific needs of a particular user community, providing a more targeted and simplified view of the data.
    c) They are typically created through the extraction and filtering of relevant data from the data warehouse into a separate database.
8) **Security and Governance:**
    a) Data warehouses often contain sensitive and critical data, so security and governance mechanisms are crucial.
    b) Access controls, data encryption, authentication, and authorization mechanisms are implemented to ensure data privacy and prevent unauthorized access.
    c) Data governance processes and policies are established to define data standards, ensure data quality, and enforce data usage policies.

OLAP OPERATIONS AND QUERIES

- OLAP (Online Analytical Processing) operations and queries are essential components of data analysis and reporting in the field of business intelligence.
- OLAP refers to a set of techniques and tools used to perform multidimensional analysis of data, allowing users to gain insights from complex and large datasets.
- OLAP operations and queries enable users to navigate and manipulate data cubes, which are multi-dimensional structures that organize data for efficient analysis.

There are four primary OLAP operations: drill-down, roll-up, slice-and-dice, and pivot. Each operation serves a specific purpose and aids in exploring data from different perspectives.

Drill-Down:

- The drill-down operation allows users to explore data at a more detailed level by moving from higher-level summaries to lower-level details.

- It involves expanding dimensions in a data cube to reveal additional levels of granularity.
- For example, if a user is analysing sales data by year, they can drill down to examine sales by quarter, then by month, and finally by individual days.

Roll-Up:

- Roll-up is the reverse operation of drill-down.
- It enables users to summarize data at a higher level by collapsing or aggregating data from detailed levels.
- It helps in analysing data at various levels of aggregation.
- For instance, a user can roll up sales data from the daily level to monthly, quarterly, or yearly levels to gain a broader perspective on the overall sales performance.

Slice-and-Dice:

- Slice-and-dice allows users to extract specific subsets of data from a data cube based on certain criteria.
- Slicing involves selecting a particular dimension value to view a subset of data that corresponds to that value.
- Dicing, on the other hand, involves selecting multiple dimension values simultaneously to create a more specific subset of data.
- For example, a user can slice the sales data cube to analyse sales for a specific region or dice it to analyse sales for a particular region and product category.

Pivot:

- The pivot operation rotates the data cube to provide alternative views or rearrangements of dimensions.
- It allows users to reorganize the data along different dimensions, making it easier to analyse data from various perspectives.
- By pivoting the data, users can change the primary focus of the analysis.
- For instance, a user can pivot a sales data cube to view sales by product category instead of by region.

Query language

- OLAP queries are used to retrieve specific information from a data cube based on user requirements.
- These queries are typically expressed using a specialized query language like MDX (Multidimensional Expressions) or OLAP SQL.
- OLAP queries can involve a combination of dimensions, hierarchies, measures, and filters to define the desired analysis.

- OLAP operations and queries play a crucial role in business intelligence applications, providing analysts and decision-makers with the ability to explore data from multiple dimensions and levels of detail.
- By leveraging these operations and queries, users can gain insights into trends, patterns, and anomalies in data, facilitating informed decision-making and strategic planning.

MDX (Multidimensional Expressions)

MDX (Multidimensional Expressions) is a query language specifically designed for querying and manipulating data stored in multidimensional databases or OLAP cubes. It provides a rich set of functions and operators to perform complex analysis and calculations on multidimensional data. Here are some key aspects of MDX:

1. Query Structure: MDX queries consist of various clauses, including the SELECT clause, FROM clause, WHERE clause, and optional clauses like the HAVING clause and ORDER BY clause. These clauses define the dimensions, measures, and filters for the query.

2. Dimensions and Hierarchies: MDX queries work with dimensions and hierarchies, which represent the organizational structure of the data. Dimensions are used to categorize data, while hierarchies define the levels of aggregation within a dimension.

3. Functions and Operators: MDX provides a wide range of functions and operators to perform calculations, aggregations, comparisons, and transformations on the data. These include arithmetic functions, statistical functions, time-related functions, set functions, and more.

4. Member Selection and Navigation: MDX allows users to select specific members from dimensions and navigate through hierarchies to access data at different levels of detail. Users can drill down, drill up, and cross-navigate between hierarchies to explore and analyse the data.

5. Calculated Measures and Sets: MDX supports the creation of calculated measures and sets. Calculated measures are derived measures that are not stored in the database but calculated on the fly during query execution. Calculated sets are subsets of data defined by specific criteria.

6. Advanced Analysis: MDX enables advanced analytical operations such as time series analysis, ranking, top/bottom filtering, and multidimensional data transformations. It allows users to perform complex calculations, aggregations, and comparisons across dimensions and hierarchies.

MDX is a powerful language for querying multidimensional databases and performing advanced analysis on OLAP data. It provides the flexibility and expressiveness required to analyse complex business scenarios, generate reports, and extract valuable insights from multidimensional data structures.