**OLTP (Online Transaction Processing):**

* **Purpose:** Handles real-time transactional data (e.g., banking, order processing).
* **Schema:** Highly normalized (e.g., 3rd Normal Form) to:
  + Minimize data redundancy.
  + Optimize frequent insert, update, and delete operations (DML).
* **Query Types:** Simple and fast queries for CRUD operations.
* **Workload:** High volume of concurrent, short, and fast transactions.

**OLAP (Online Analytical Processing):**

* **Purpose:** Designed for analytical queries and reporting (e.g., dashboards, trend analysis).
* **Schema:**
  + Dimensional modeling is common (e.g., star or snowflake schema) for optimized querying.
  + Can also use 3NF, but less common due to query performance.
* **Query Types:** Complex read-heavy queries (e.g., aggregations, joins).
* **Workload:** Low volume of long-running, complex queries.

**How Data Modeling Differs:**

1. **OLTP Data Modeling:**
   * Focuses on **normalization** to reduce redundancy and ensure data integrity.
   * Common models: ERD (Entity-Relationship Diagram) with relationships between entities.
2. **OLAP Data Modeling:**
   * Focuses on **denormalization** for fast query performance.
   * Common models: Dimensional models (star schema, snowflake schema).

**Example to Illustrate:**

* **OLTP System (E-commerce Order System):**
  + Tables: Customers, Orders, Products, Inventory.
  + Schema: Normalized for efficient updates and real-time transactions.
* **OLAP System (E-commerce Sales Analysis):**
  + Tables: Fact table for Sales, Dimension tables for Customers, Products, Time.
  + Schema: Star schema for quick aggregation and reporting.

Question 2: Explain the differences between a star schema and a snowflake schema. When would you use one over the other?

**Key Points to Highlight:**

**Star Schema:**

* **Structure:**
  + Consists of a **central fact table** connected directly to **dimension tables**.
  + Dimension tables are **denormalized** (flattened) with attributes stored in a single table.
* **Performance:**
  + Optimized for **fast query performance** as it minimizes the number of joins.
  + Suitable for **read-heavy analytical workloads**.
* **Use Case:**
  + Preferred when query performance is critical, and storage is not a major concern.
  + Works well for simpler business use cases with fewer dimensions.

**Snowflake Schema:**

* **Structure:**
  + Consists of a **central fact table** connected to **normalized dimension tables**.
  + Dimension tables are further broken down into sub-dimension tables to eliminate redundancy.
* **Performance:**
  + Requires **more joins** for queries, which can impact performance for large datasets.
  + Optimized for **storage efficiency** and reducing redundancy.
* **Use Case:**
  + Useful when storage optimization is prioritized or when dimensions have complex relationships.
  + Works well for complex and larger datasets with hierarchical relationships.

**Comparison Table:**

| **Feature** | **Star Schema** | **Snowflake Schema** |
| --- | --- | --- |
| **Normalization** | Denormalized dimensions | Normalized dimensions |
| **Joins** | Fewer joins (direct dimensions) | More joins (normalized structure) |
| **Performance** | Faster query performance | Slower due to joins |
| **Storage** | Redundant data (higher storage) | Optimized storage (less redundancy) |
| **Complexity** | Simpler structure | More complex structure |

**Example Use Case:**

**Star Schema:**

* **Sales Reporting System:**
  + Central fact table: Sales (e.g., Date, ProductID, CustomerID, SalesAmount).
  + Dimensions: Products, Customers, Time (all flattened for quick access).

**Snowflake Schema:**

* **Retail Analytics System:**
  + Central fact table: Sales.
  + Dimensions:
    - Products (normalized into ProductDetails, ProductCategory).
    - Customers (normalized into CustomerDetails, CustomerLocation).

**Summary:**

* **Choose Star Schema** when: Query performance is critical, data redundancy is acceptable, and the use case is relatively straightforward.
* **Choose Snowflake Schema** when: Storage efficiency is crucial, data is highly normalized, and the dataset involves complex hierarchies.

Question 3: **What are fact and dimension tables in data modeling? How do they relate to each other?**

**Key Points to Highlight:**

**Fact Table:**

* **Definition:**
  + A fact table stores **numerical, quantitative data** that represents **business events** or **metrics** (e.g., sales, revenue, inventory levels).
* **Key Features:**
  + Contains **measures** (e.g., SalesAmount, Quantity, Profit).
  + Contains **foreign keys** that link to the related dimension tables.
  + Typically grows much larger than dimension tables as it captures business transactions.
* **Example:**
  + **Fact Table: Sales**

| **SaleID** | **ProductID** | **CustomerID** | **DateID** | **SalesAmount** | **Quantity** |
| --- | --- | --- | --- | --- | --- |

**Dimension Table:**

* **Definition:**
  + A dimension table contains **descriptive attributes** that provide context to the measures in the fact table.
* **Key Features:**
  + Stores textual or categorical information (e.g., CustomerName, ProductCategory, Date).
  + Contains a **primary key** used to join with the fact table.
  + Typically smaller and changes less frequently than fact tables.
* **Example:**
  + **Dimension Table: Products**

| **ProductID** | **ProductName** | **Category** | **Manufacturer** |
| --- | --- | --- | --- |

**Relationship Between Fact and Dimension Tables:**

* Fact tables and dimension tables are connected using **primary keys (PK)** in the dimension table and **foreign keys (FK)** in the fact table.
* Dimension tables help provide the context needed to **slice and dice** the numerical data in the fact table.
* These relationships are the backbone of **star** and **snowflake schemas**.

**Key Differences:**

| **Feature** | **Fact Table** | **Dimension Table** |
| --- | --- | --- |
| **Data Type** | Numerical (measures) | Descriptive (attributes) |
| **Purpose** | Stores business metrics | Provides context for metrics |
| **Size** | Large (grows with transactions) | Smaller (static or slowly changing) |
| **Keys** | Contains foreign keys | Contains primary keys |
| **Examples** | SalesAmount, Quantity | CustomerName, ProductCategory |

**Example Use Case:**

**Fact Table: Sales**

| **SaleID** | **ProductID** | **CustomerID** | **DateID** | **SalesAmount** | **Quantity** |
| --- | --- | --- | --- | --- | --- |

**Dimension Table: Customers**

| **CustomerID** | **CustomerName** | **Location** | **AgeGroup** |
| --- | --- | --- | --- |

**How They Relate:**

* Fact table (Sales) contains CustomerID as a **foreign key**.
* Dimension table (Customers) contains CustomerID as a **primary key**.
* These allow analytics like "Total sales by location" or "Top customers by age group."

**Summary:**

* **Fact Tables:** Quantitative data (e.g., sales, revenue) for analysis.
* **Dimension Tables:** Qualitative attributes (e.g., product details, customer demographics) to provide context.
* They work together to support analytical use cases and enable slicing and dicing of data.

Question 4: **What is slowly changing dimension (SCD)? Explain the types with examples.**

**Slowly Changing Dimensions (SCD):**

* **Definition:**
  + SCD is a technique used in data modeling to handle and track changes in dimension tables over time.
  + Changes in dimensions (like customer address or product price) need to be captured while balancing performance, storage, and query requirements.
* **Why It’s Important:**
  + Helps maintain historical context in analytical systems.
  + Supports accurate reporting and trend analysis.

**Types of SCD:**

**1. SCD Type 1: Overwrite**

* **Description:**
  + The old data is overwritten with the new data.
  + No history of changes is retained.
* **Use Case:**
  + Useful for non-critical changes like correcting a typo or when history isn’t required.
* **Example:**
  + Customer table: Updating a customer’s phone number. | CustomerID | Name | PhoneNumber | |------------|--------|-------------| | 1 | Alice | 1234567890 |
  + After an update: | CustomerID | Name | PhoneNumber | |------------|--------|-------------| | 1 | Alice | 0987654321 |

**2. SCD Type 2: Full History**

* **Description:**
  + Tracks historical changes by adding new rows for each change.
  + Uses flags, version numbers, or effective dates to indicate the current and historical rows.
* **Use Case:**
  + Critical for tracking complete history, such as a customer moving to a new location or a product price change.
* **Example:**
  + Customer table: Tracking address changes. | CustomerID | Name | Address | IsCurrent | EffectiveFrom | EffectiveTo | |------------|--------|----------------|-----------|---------------|---------------| | 1 | Alice | 123 Main St | Y | 2023-01-01 | NULL |
  + After an address change: | CustomerID | Name | Address | IsCurrent | EffectiveFrom | EffectiveTo | |------------|--------|----------------|-----------|---------------|---------------| | 1 | Alice | 123 Main St | N | 2023-01-01 | 2023-05-01 | | 1 | Alice | 456 Elm St | Y | 2023-05-01 | NULL |

**3. SCD Type 3: Limited History**

* **Description:**
  + Tracks only a limited amount of history (e.g., the current and one previous state).
  + Adds columns to store old values (e.g., PreviousAddress).
* **Use Case:**
  + When storage is limited, and full history isn’t required.
* **Example:**
  + Customer table: Tracking one previous address. | CustomerID | Name | CurrentAddress | PreviousAddress | |------------|--------|----------------|-----------------| | 1 | Alice | 456 Elm St | 123 Main St |

**Summary of SCD Types:**

| **Type** | **Tracks History?** | **How It Works** | **Use Case** |
| --- | --- | --- | --- |
| **SCD 1** | No | Overwrites existing data | Non-critical changes |
| **SCD 2** | Yes (Full) | Adds new rows for each change | Critical historical tracking |
| **SCD 3** | Yes (Limited) | Adds columns to store limited historical data | Limited history with space concerns |

**Which Type to Use:**

* **SCD 1:** Use when maintaining history isn’t required (e.g., correcting typos).
* **SCD 2:** Use when tracking full history is essential for analysis or compliance.
* **SCD 3:** Use when partial history is enough, and storage or simplicity is a priority.

Question 5: **What is a surrogate key, and why is it used in data modeling? How does it differ from a natural key?**

**Surrogate Key:**

* **Definition:**
  + A surrogate key is an **artificially generated unique identifier** for a record in a table.
  + It is not derived from any application data and has no business meaning.
* **Why It’s Used:**
  + Ensures uniqueness in cases where a natural key is not available or practical.
  + Helps handle scenarios like duplicate data, changes in natural keys, or complex composite keys.
* **Characteristics:**
  + Typically implemented as an integer or UUID.
  + Automatically generated (e.g., auto-increment in databases).
  + Independent of application logic or real-world changes.

**Natural Key:**

* **Definition:**
  + A natural key is a **unique identifier derived from real-world data** that represents the record (e.g., Social Security Number, ISBN, or email address).
* **Why It’s Used:**
  + Directly ties the data to its real-world entity.
  + Simplifies data access for certain applications.
* **Challenges:**
  + May not always guarantee uniqueness (e.g., two people can share the same name).
  + Changes in the natural key (e.g., a customer changes their email address) can complicate data updates.

**Key Differences:**

| **Aspect** | **Surrogate Key** | **Natural Key** |
| --- | --- | --- |
| **Definition** | Artificial identifier with no business meaning | Real-world attribute representing the entity |
| **Uniqueness** | Always unique | May not always guarantee uniqueness |
| **Change Impact** | Independent of real-world changes | Changes in real-world data affect the key |
| **Complexity** | Simple and auto-generated | Can be complex, especially for composite keys |
| **Example** | Auto-increment ID (e.g., 101, 102) | Email address, ISBN, Social Security Number |

**Example:**

**Natural Key:**

* **Table: Customers**

| **Email** | **Name** | **Address** |
| --- | --- | --- |
| alice@mail.com | Alice | 123 Main St |
| bob@mail.com | Bob | 456 Elm St |

* + Email serves as the natural key.

**Surrogate Key:**

* **Table: Customers**

| **CustomerID** | **Email** | **Name** | **Address** |
| --- | --- | --- | --- |
| 101 | alice@mail.com | Alice | 123 Main St |
| 102 | bob@mail.com | Bob | 456 Elm St |

* + CustomerID is the surrogate key.

**Why Use Surrogate Keys:**

1. **Simplifies Joins:** Surrogate keys (e.g., integers) are easier to work with than composite natural keys in joins.
2. **Handles Changes:** Natural keys like Email may change, breaking referential integrity. Surrogate keys avoid this problem.
3. **Avoids Complexity:** Surrogate keys prevent using complex composite natural keys (e.g., multiple columns).

**When to Use Natural Keys:**

* Use a natural key when:
  + The natural key is truly unique and unlikely to change (e.g., Social Security Number).
  + It directly reflects the business domain and simplifies application logic.

**Question 6: What is normalization in data modeling? Explain the different normal forms with examples.**

**Normalization:**

* **Definition:**
  + A process in database design to organize data, reduce redundancy, and improve data integrity.
  + Normalization ensures data dependencies are logical and prevents anomalies (insertion, update, and deletion anomalies).

**1NF: First Normal Form**

* **Rule:**
  + Each cell contains a single, atomic value (no repeating groups or arrays).
* **Example:**
  + **Unnormalized Table:**

| **StudentID** | **Name** | **Subjects** |
| --- | --- | --- |
| 1 | Alice | Math, Science |
| 2 | Bob | Math, English |

* + **Normalized to 1NF:**

| **StudentID** | **Name** | **Subject** |
| --- | --- | --- |
| 1 | Alice | Math |
| 1 | Alice | Science |
| 2 | Bob | Math |
| 2 | Bob | English |

**2NF: Second Normal Form**

* **Rule:**
  + Must satisfy **1NF**.
  + No **partial dependency**: Non-key attributes must depend on the **entire primary key**, not just a part of it (applies to tables with composite keys).
* **Example:**
  + **1NF Table:**

| **StudentID** | **CourseID** | **CourseName** | **Instructor** |
| --- | --- | --- | --- |
| 1 | C101 | Math | Mr. Smith |
| 2 | C102 | Science | Dr. Adams |

* + - **Issue:** CourseName and Instructor depend only on CourseID (not the entire composite key StudentID, CourseID).
  + **Normalized to 2NF:**
    - Split the table into two:
    - **Student-Course Table:**

| **StudentID** | **CourseID** |
| --- | --- |
| 1 | C101 |
| 2 | C102 |

* + - **Course Table:**

| **CourseID** | **CourseName** | **Instructor** |
| --- | --- | --- |
| C101 | Math | Mr. Smith |
| C102 | Science | Dr. Adams |

**3NF: Third Normal Form**

* **Rule:**
  + Must satisfy **2NF**.
  + No **transitive dependency**: Non-key attributes must depend only on the primary key, not on other non-key attributes.
* **Example:**
  + **2NF Table:**

| **CourseID** | **CourseName** | **Instructor** | **InstructorPhone** |
| --- | --- | --- | --- |
| C101 | Math | Mr. Smith | 1234567890 |
| C102 | Science | Dr. Adams | 0987654321 |

* + - **Issue:** InstructorPhone depends on Instructor, not directly on CourseID.
  + **Normalized to 3NF:**
    - Split the table into two:
    - **Course Table:**

| **CourseID** | **CourseName** | **Instructor** |
| --- | --- | --- |
| C101 | Math | Mr. Smith |
| C102 | Science | Dr. Adams |

* + - **Instructor Table:**

| **Instructor** | **InstructorPhone** |
| --- | --- |
| Mr. Smith | 1234567890 |
| Dr. Adams | 0987654321 |

**Summary of Normal Forms:**

| **Normal Form** | **Rule** | **Purpose** |
| --- | --- | --- |
| **1NF** | Atomic values, no repeating groups | Eliminate repeating groups. |
| **2NF** | 1NF + no partial dependencies | Eliminate redundancy in composite keys. |
| **3NF** | 2NF + no transitive dependencies | Eliminate indirect dependencies. |

Question 7: **What is denormalization in data modeling? When would you use it?**

**Denormalization:**

* **Definition:**
  + Denormalization is the process of **introducing redundancy** into a database design by combining tables or precomputing data.
  + It is the opposite of normalization and is often used to **optimize query performance** for read-heavy systems like OLAP.

**Key Characteristics:**

1. **Data Redundancy:**
   * Duplicate data is intentionally stored to reduce the complexity of joins.
2. **Performance Trade-off:**
   * Improves read performance (e.g., fewer joins) but may degrade write performance and increase storage requirements.
3. **Storage Impact:**
   * Requires more storage space due to redundancy.

**When to Use Denormalization:**

1. **OLAP Systems:**
   * Analytical systems prioritize query performance over storage efficiency (e.g., dashboards, reporting).
2. **Frequent Joins:**
   * When frequent joins between normalized tables degrade query performance.
3. **Precomputed Aggregations:**
   * To reduce query execution time for common aggregations (e.g., total sales by category).
4. **Read-Heavy Workloads:**
   * Systems with a high volume of read operations benefit from denormalization.

**Example:**

**Normalized Schema:**

**Tables:**

1. **Orders**

| **OrderID** | **CustomerID** | **ProductID** | **Quantity** |
| --- | --- | --- | --- |
| 101 | 1 | P1 | 2 |
| 102 | 2 | P2 | 3 |

1. **Products**

| **ProductID** | **ProductName** | **Category** |
| --- | --- | --- |
| P1 | Laptop | Electronics |
| P2 | Chair | Furniture |

1. **Customers**

| **CustomerID** | **CustomerName** | **City** |
| --- | --- | --- |
| 1 | Alice | NY |
| 2 | Bob | LA |

**Denormalized Schema:**

* Combine tables into a single table for faster querying.

**Denormalized Table:**

| **OrderID** | **CustomerName** | **ProductName** | **Category** | **Quantity** | **City** |
| --- | --- | --- | --- | --- | --- |
| 101 | Alice | Laptop | Electronics | 2 | NY |
| 102 | Bob | Chair | Furniture | 3 | LA |

**Advantages of Denormalization:**

1. **Improved Read Performance:**
   * Fewer joins and reduced query complexity.
2. **Faster Aggregation:**
   * Precomputed data (e.g., totals) eliminates the need for on-the-fly calculations.

**Disadvantages of Denormalization:**

1. **Increased Storage:**
   * Redundant data consumes more space.
2. **Data Inconsistency:**
   * Changes in one place may require updates in multiple locations.
3. **Slower Write Performance:**
   * Inserts, updates, and deletes may require more time due to redundancy.

**Use Case:**

* **Data Warehouse:**
  + Use a denormalized star schema with fact and dimension tables for faster query performance.
* **High-Performance Reports:**
  + Precompute metrics like sales totals in denormalized tables to avoid expensive runtime calculations.

**Summary:**

* **Normalization:** Reduces redundancy, optimizes write operations, and ensures data integrity.
* **Denormalization:** Introduces redundancy to optimize read performance, particularly in analytical or reporting systems.

**Data Warehouse Basics**

**1. What is a data warehouse, and how does it differ from a database?**

* **Data Warehouse:**
  + A central repository optimized for **analytical queries** and **reporting**.
  + Stores **historical data** from multiple sources.
  + Structured for **read-heavy workloads**.
* **Database:**
  + Designed for **transactional operations** (OLTP).
  + Stores **current data** for day-to-day operations.
  + Optimized for **write-heavy workloads**.
* **Example:**
  + **Database:** Processes customer orders (OLTP).
  + **Data Warehouse:** Aggregates and analyzes sales trends (OLAP).

**2. Explain the key components of a data warehouse architecture.**

* **Staging Area:**
  + Temporary storage for raw data extracted from source systems.
* **Integration Layer:**
  + Cleansed, transformed data stored in relational format (3NF or star schema).
* **Presentation Layer:**
  + Optimized data for analytics (denormalized, dimensional modeling).
* **Metadata Repository:**
  + Stores information about the data (e.g., lineage, structure).
* **Tools:** ETL (e.g., Informatica, Talend), BI Tools (e.g., Power BI, Tableau).

**3. What is the difference between ETL and ELT?**

* **ETL:** Extract → Transform → Load
  + Transformation happens before loading data.
  + Suitable for traditional on-prem data warehouses.
* **ELT:** Extract → Load → Transform
  + Transformation happens inside the target system (e.g., cloud data warehouse).
  + Faster with modern tools like Snowflake or BigQuery.

**4. What are the benefits of using a data warehouse in an organization?**

* **Centralized Data:** Unified view of data from multiple sources.
* **Historical Analysis:** Enables trend analysis over time.
* **Improved Decision-Making:** Supports advanced analytics and reporting.
* **Performance Optimization:** Faster query performance for analytics.

**Schemas and Modeling**

**5. What is a star schema? How does it differ from a snowflake schema?**

* **Star Schema:**
  + Central fact table connected directly to denormalized dimension tables.
  + Fewer joins → Faster queries.
* **Snowflake Schema:**
  + Central fact table connected to normalized dimension tables.
  + Saves storage but requires more joins.
* **Example:**
  + **Star:** Sales → Customers, Products (denormalized).
  + **Snowflake:** Sales → Customers → Regions (normalized).

**6. Explain the concept of fact and dimension tables with examples.**

* **Fact Table:**
  + Stores numerical metrics (e.g., sales, revenue).
  + Contains foreign keys to dimensions.
* **Dimension Table:**
  + Stores descriptive attributes (e.g., customer name, product category).
  + Contains primary keys to join with facts.
* **Example:**
  + **Fact Table:**  
    | SaleID | ProductID | CustomerID | Revenue |
  + **Dimension Table:**  
    | CustomerID | CustomerName | Location |

**7. What is a surrogate key, and why is it preferred over a natural key in data warehouses?**

* **Surrogate Key:**
  + An artificial, unique key (e.g., auto-increment ID).
  + Independent of business logic.
* **Why Preferred:**
  + Handles changes in natural keys (e.g., email updates).
  + Simplifies joins (e.g., single integer key vs composite keys).

**8. What are additive, semi-additive, and non-additive facts?**

* **Additive:** Summable across all dimensions (e.g., sales amount).
* **Semi-Additive:** Summable across some dimensions (e.g., account balance over time).
* **Non-Additive:** Not summable (e.g., percentages).

**Normalization and Denormalization**

**9. What is normalization? Explain the different normal forms with examples.**

* **Normalization:**
  + Organizes data to reduce redundancy.
  + **1NF:** Atomic values (no repeating groups).
  + **2NF:** 1NF + no partial dependencies.
  + **3NF:** 2NF + no transitive dependencies.

**10. What is denormalization, and when would you use it in a data warehouse?**

* **Denormalization:**
  + Introduces redundancy to improve query performance.
  + **Use Case:** Analytical workloads (OLAP).

**Slowly Changing Dimensions (SCDs)**

**11. What is a slowly changing dimension (SCD)?**

* **Definition:**
  + Tracks changes in dimension data over time.
  + **Type 1:** Overwrites data.
  + **Type 2:** Maintains full history.
  + **Type 3:** Tracks limited history.

**12. How do you implement SCD Type 2 in ETL pipelines?**

* Compare incoming data with the current dimension.
* Insert new rows for changes, with EffectiveFrom and EffectiveTo columns.

**Data Modeling Approaches**

**13. Compare and contrast the Kimball and Inmon methodologies.**

* **Kimball:** Bottom-up, uses dimensional modeling.
* **Inmon:** Top-down, uses 3NF in data warehouse.

**14. What is data vault modeling?**

* **Definition:**
  + Hub (business keys), satellites (descriptive data), and links (relationships).
* **Use Case:** Real-time data integration.

**15. How do you handle hierarchies in a dimensional model?**

* Use self-referencing tables or snowflake schema (e.g., Region → Country → City).

**Granularity and Performance**

**16. What is the granularity of a fact table?**

* Level of detail stored (e.g., daily sales vs. hourly sales).

**17. How would you optimize query performance?**

* Indexing, partitioning, materialized views, and caching.

**18. What is indexing, and how can it be applied?**

* **Index:** Speeds up data retrieval by creating pointers.
* Use on foreign keys and commonly queried columns.

**19. How does partitioning improve query performance?**

* Divides large tables into smaller chunks (e.g., by date or region).

**Advanced Concepts**

**20. What are conformed dimensions?**

* Shared dimensions across multiple fact tables (e.g., shared Customer dimension in sales and returns).

**21. Explain transactional and snapshot fact tables.**

* **Transactional:** One row per event (e.g., sales).
* **Snapshot:** Summarized data at regular intervals (e.g., daily inventory).

**22. What is a late-arriving dimension?**

* Data arriving after the fact data. Use default surrogate keys and update later.

**23. How would you design a real-time data warehouse?**

* Use stream processing tools (e.g., Kafka) with incremental ETL.

**Case Study Questions**

**24. Design a data warehouse for an e-commerce company. Include fact and dimension tables.**

**Scenario:**

The data warehouse should track sales, customers, products, and store details.

**Schema Design:**

1. **Fact Table: Sales**
   * Captures sales transactions.
   * **Columns:**
     + SaleID (PK), ProductID (FK), CustomerID (FK), StoreID (FK), DateID (FK), Quantity, SalesAmount.

| **SaleID** | **ProductID** | **CustomerID** | **StoreID** | **DateID** | **Quantity** | **SalesAmount** |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | P101 | C001 | S001 | 20230101 | 3 | 90.00 |

1. **Dimension Tables:**
   * **Products:** Stores product details.

| **ProductID** | **ProductName** | **Category** | **Price** |
| --- | --- | --- | --- |
| P101 | Laptop | Electronics | 30.00 |

* + **Customers:** Stores customer details.

| **CustomerID** | **CustomerName** | **Region** |
| --- | --- | --- |
| C001 | Alice | North |

* + **Stores:** Stores details of physical locations.

| **StoreID** | **StoreName** | **City** |
| --- | --- | --- |
| S001 | MainStore | New York |

* + **Time:** Tracks date-related attributes.

| **DateID** | **Date** | **Month** | **Year** |
| --- | --- | --- | --- |
| 20230101 | 01-01-2023 | Jan | 2023 |

**25. Sketch a schema for a sales reporting system that tracks daily, weekly, and monthly trends.**

**Fact Table: Sales**

* Stores daily sales data.
* **Columns:**
  + SaleID, ProductID (FK), DateID (FK), Quantity, SalesAmount.

**Dimension Tables:**

1. **Date Dimension:**
   * Tracks daily, weekly, and monthly trends.
   * **Columns:** DateID, Date, WeekNumber, Month, Quarter, Year.
2. **Product Dimension:**
   * Tracks product details like ProductName, Category.
3. **Store Dimension:**
   * Tracks sales location.

**Star Schema Diagram:**

sql

Copy code

+------------+

| Date |

+------------+

|

|

+---------+ +---------+ +---------+

| Product |---| Sales |---| Store |

+---------+ +---------+ +---------+

**26. Explain how you would design a data warehouse for a healthcare system to track patient visits and treatment history.**

**Scenario:**

Track patient visits, diagnoses, treatments, and associated medical staff.

**Schema Design:**

1. **Fact Table: Visits**
   * Captures visit-specific data.
   * **Columns:** VisitID, PatientID (FK), DoctorID (FK), TreatmentID (FK), DateID (FK), Cost.

| **VisitID** | **PatientID** | **DoctorID** | **TreatmentID** | **DateID** | **Cost** |
| --- | --- | --- | --- | --- | --- |
| 1 | P001 | D001 | T001 | 20230101 | 200.00 |

1. **Dimension Tables:**
   * **Patients:** Stores patient demographics.

| **PatientID** | **PatientName** | **Age** | **Gender** |
| --- | --- | --- | --- |
| P001 | John Doe | 30 | Male |

* + **Doctors:** Stores doctor information.

| **DoctorID** | **DoctorName** | **Specialization** |
| --- | --- | --- |
| D001 | Dr. Smith | Cardiology |

* + **Treatments:** Stores treatment details.

| **TreatmentID** | **TreatmentName** | **Cost** |
| --- | --- | --- |
| T001 | ECG | 200 |

* + **Time Dimension:** Tracks visit date and time attributes.

**Practical Questions**

**27. How would you handle a dimension that changes frequently, such as product pricing?**

* Use **SCD Type 2** to maintain history for price changes:
  + **Approach:**
    - Add EffectiveFrom and EffectiveTo columns.
    - Insert a new row for each price change.
  + **Example:**

| **ProductID** | **ProductName** | **Price** | **EffectiveFrom** | **EffectiveTo** |
| --- | --- | --- | --- | --- |
| P101 | Laptop | 30.00 | 2023-01-01 | 2023-03-01 |
| P101 | Laptop | 35.00 | 2023-03-02 | NULL |

**28. How do you deal with null values or missing data in a data warehouse?**

* **Approaches:**
  + **Default Values:** Use default values (e.g., Unknown for text fields).
  + **Imputation:** Replace nulls with averages, medians, or other estimations.
  + **Separate Rows:** Store missing data in a separate table for auditing.

**29. What are surrogate keys, and how do you generate them in an ETL pipeline?**

* **Definition:** Artificial unique identifiers for dimension tables.
* **Generation Methods:**
  + Auto-increment columns in databases.
  + UUIDs or hash-based IDs in ETL tools.
  + **Example in SQL:**

sql

Copy code

INSERT INTO Customers (CustomerID, CustomerName)

SELECT ROW\_NUMBER() OVER (ORDER BY CustomerName), CustomerName

FROM StagingCustomers;

**30. How do you test the accuracy and integrity of your data warehouse?**

* **Techniques:**
  + **Row Count Validation:** Verify that record counts match between source and destination.
  + **Data Completeness:** Ensure no data is lost or truncated during ETL.
  + **Referential Integrity Checks:** Ensure foreign keys match primary keys.
  + **Business Rule Testing:** Validate against expected outcomes (e.g., total sales = sum of individual transactions).
  + **Example:**
    - Validate total revenue:

sql

Copy code

SELECT SUM(SalesAmount) FROM Sales WHERE Date BETWEEN '2023-01-01' AND '2023-01-31';