# Designing Efficient Database Systems with Schemas

As a business intelligence (BI) professional, understanding various database schemas is essential for organizing and optimizing data storage. This reading will explore common schema types you'll encounter, highlighting their structures and uses.

**Types of Schemas**

**Star Schema**

The **star schema** consists of a central fact table linked to multiple dimension tables, resembling a star. This structure is designed for efficient data retrieval and is particularly useful for high-scale information delivery.

**Key Features:**

- **Fact Table**: Contains quantitative data for analysis (e.g., sales).
- **Dimension Tables**: Provide context (e.g., customer info, product details).
- **Read Efficiency**: Optimized for read operations, making it ideal for reporting and data analysis.

**Snowflake Schema**

The **snowflake schema** is a more complex variation of the star schema. It includes normalized dimension tables, breaking them into additional sub-dimensions, which create a snowflake-like structure.

**Key Features:**

- **Normalization**: Reduces data redundancy by organizing data into multiple related tables.
- **Complexity**: Can accommodate more detailed relationships among data.

**Flat Model**

**Flat models** consist of a single table where each record corresponds to a row, and columns separate fields. These schemas are non-relational and ideal for simple, static data.

**Key Features:**

- **Simplicity**: Easy to create and manage.
- **No Relationships**: Cannot capture relationships between data items.

**Semi-Structured Schemas**

**Semi-structured schemas** offer flexibility while maintaining some organization. These schemas allow for varied data types and structures, making them adaptable for diverse data sets.

# Database Comparison Checklist

Understanding the various types of databases is crucial for business intelligence (BI) professionals as it directly influences how data is processed and utilized. Here's a comprehensive checklist comparing key database types based on different aspects.

**OLAP vs. OLTP**

| Database Technology | Description | Use |
|---|---|---|
| **OLAP** | Online Analytical Processing systems optimized for data analysis. | Provides access to data from multiple sources; used for BI and decision-making. Analyzes data for actionable insights from reporting tables. |
| **OLTP** | Online Transaction Processing systems optimized for transaction processing. | Stores transaction data for customer-facing applications; reads, writes, and updates single rows of data. Acts as source systems for data pipelines. |

**Row-Based vs. Columnar**

| Database Technology | Description | Use |
|---|---|---|
| **Row-Based** | Databases organized by rows, commonly used in OLTP systems. | Fast data writing; stores all values of a row together; easily optimized with indexing. |
| **Columnar** | Databases organized by columns, typically used for OLAP systems. | Faster data reading; pulls only necessary data for analysis; stores multiple rows' columns together. |

**Distributed vs. Single-Homed**

| Database Technology | Description | Use |
|---|---|---|
| **Distributed** | Collections of data systems located across multiple physical locations. | Easily expandable for larger business needs; accessed from different networks; generally more secure. |
| **Single-Homed** | Databases where all data is stored in a single physical location. | Easier data access and coordination; reduces redundancy; cost-effective to maintain. |

**Separated Storage and Compute vs. Combined**

| Database Technology | Description | Use |
|---|---|---|
| **Separated Storage and Compute** | Systems where less relevant data is stored remotely and relevant data is local. | Allows for efficient analytical queries by processing only necessary data; enables scaling of resources independently. |
| **Combined Storage and Compute** | Systems that store and analyze data in the same place. | Traditional setup for straightforward resource management; provides access to all data simultaneously. |

# Four Key Elements of Database Schemas

Whether you are creating a new database model or exploring a system in place already, it is important to ensure that all elements exist in the schema. The database schema enables you to validate incoming data being delivered to your destination database to prevent errors and ensure the data is immediately useful to users.

Here is a checklist of common elements a database schema should include:

- **The relevant data:** The schema describes how the data is modelled and shaped within the database and must encompass all of the data being described.
- **Names and data types for each column:** Include names and data types for each column in each table within the database.
- **Consistent formatting:** Ensure consistent formatting across all data entries. Every entry is an instance of the schema, so it needs to be consistent.
- **Unique keys:** The schema must use unique keys for each entry within the database. These keys build connections between the tables and enable users to combine relevant data from across the entire database.

# ETL vs. ELT: Key Differences and Considerations

Understanding the differences between ETL (Extract, Transform, Load) and ELT (Extract, Load, Transform) is essential for optimizing data pipelines and ensuring efficient data processing. Here's a detailed comparison to help you decide which approach best suits your organization's needs.

**Key Differences**

| Aspect | ETL | ELT |
|---|---|---|
| **Order of Operations** | Extract, Transform, Load | Extract, Load, Transform |
| **Location of Transformations** | Transformations occur in a staging area before loading | Transformations occur in the destination system after loading |
| **Age of Technology** | Established for over 20 years, with many supporting tools | Newer technology with fewer built-in support tools |
| **Data Access** | Transforms and loads only designated data | Loads all data, allowing on-demand analysis of any subset |
| **Calculations** | Replaces or revises existing columns in the target table | Adds calculations directly to the existing dataset |
| **Compatible Storage Systems** | Typically integrated with structured, relational data warehouses | Can ingest unstructured data from sources like data lakes |
| **Security and Compliance** | Sensitive data can be redacted before loading | Data must be uploaded before anonymization, increasing vulnerability |
| **Data Size Suitability** | Best for smaller datasets requiring complex transformations | Ideal for large datasets with structured and unstructured data |
| **Wait Times** | Longer load times; analysis is faster since data is pre-transformed | Faster loading; analysis may be slower as transformations occur post-load |

**Data Storage Systems**

**Data Warehouse vs. Data Lake**

Both ETL and ELT work with different data storage systems. Here's how data warehouses and data lakes compare:

| Feature | Data Warehouse | Data Lake |
|---|---|---|
| **Data Processing** | Data is processed and stored in a relational format | Data is raw and unprocessed, stored in original format |
| **Data Purpose** | Purpose of data is predefined and currently in use | Purpose is undetermined until data is analyzed |
| **System Changes** | Changes can be complex and resource-intensive | Highly accessible and easy to update |
| **Data Structure** | Structured and optimized for quick access | Unstructured, flexible storage for various data types |

**Data Marts**

- **Data Mart**: A smaller subset of a data warehouse focused on a specific subject area. It retains similar design principles but is streamlined for targeted analysis.

# A Guide to the Five Factors of Database Performance

As a Business Intelligence (BI) professional, understanding database performance is crucial for ensuring that stakeholders have quick access to the data they need for informed decision-making. Here's an overview of the five key factors that influence database performance: workload, throughput, resources, optimization, and contention.

## The Five Factors

| Factor | Definition | Example |
|---|---|---|
| **Workload** | The total volume of transactions, queries, data analyses, and commands processed by the database at any given time. | The database processes daily sales reports, performs revenue calculations, and responds to real-time stakeholder requests. |
| **Throughput** | The system's capability to process requests, influenced by input/output speed, CPU speed, and software efficiency. | Throughput encompasses the speed of data retrieval, parallel processing abilities, and the efficiency of the database management system. |
| **Resources** | The hardware and software tools available to support database operations. | A cloud-based database system that relies on online resources and software to maintain functionality. |
| **Optimization** | The process of maximizing the speed and efficiency of data retrieval to enhance database performance. | Regular checks and adjustments to ensure the database operates at peak efficiency, minimizing delays in data access. |
| **Contention** | Occurs when multiple components attempt to use the same resource simultaneously, leading to conflicts. | Automatic report generation and user-requests may conflict if they try to access the same dataset at the same time, causing slowdowns. |

# Optimization for Data Reading

Optimizing how a database reads data is crucial for enhancing performance and ensuring users have quick access to the information they need. Here are several strategies to optimize database reading, including indexing, partitioning, query optimization, and caching.

**Indexes**

Indexes in databases function similarly to an index in a book. They allow for faster data retrieval by organizing keys from database tables. Instead of scanning the entire database, an index points directly to the specific data locations, dramatically speeding up search queries.

**Partitions**

Data partitioning helps to manage and speed up database retrieval. There are two main types of partitioning:

1. **Horizontal Partitioning**: This involves dividing a table into smaller, more manageable tables based on logical groupings of rows. Each partition contains a subset of the rows, which makes querying more efficient.
2. **Vertical Partitioning**: This organizes columns into different tables. While less common, it can also optimize performance based on specific use cases.

**Other Optimization Methods**

**Queries**

Optimizing your queries can significantly reduce the time it takes to retrieve data:

- Focus on Business Requirements: Only request the data necessary for your analysis to avoid unnecessary load.
- Avoid SELECT * and SELECT DISTINCT
- Use INNER JOIN Instead of Subqueries: Joins are often more efficient than subqueries, as they process data in a more streamlined manner.
- Pre-Aggregated Queries: Store frequently accessed metrics in pre-aggregated form to avoid recalculating them with each query.

**Caching**

Caching is a valuable technique for enhancing database performance. It stores frequently accessed data in memory, allowing quicker retrieval.

# Seven elements of quality testing

In this part of the course, you have been learning about the importance of quality testing in your ETL system. This is the process of checking data for defects in order to prevent system failures. Ideally, your pipeline should have checkpoints built-in that identify any defects before they arrive in the target database system. These checkpoints ensure that the data coming in is already clean and useful!

Checking for data quality involves ensuring the data is trustworthy before reaching its destination. Ensure the quality of your data as it moves through the pipeline, there are seven elements you should consider:

- **Completeness:** Does the data contain all of the desired components or measures?
- **Consistency:** Is the data compatible and in agreement across all systems?
- **Conformity:** Does the data fit the required destination format?
- **Accuracy:** Does the data conform to the actual entity being measured or described?
- **Redundancy:** Is only the necessary data being moved, transformed, and stored for use?
- **Timeliness:** Is the data current?
- **Integrity:** Is the data accurate, complete, consistent, and trustworthy? (Integrity is influenced by the previously mentioned qualities.) Quantitative validations, including checking for duplicates, the number of records, and the amounts listed, help ensure data's integrity.

**Common issues**

There are also some common issues you can protect against within your system to ensure the incoming data doesn't cause errors or other large-scale problems in your database system:

- **Check data mapping:** Does the data from the source match the data in the target database?
- **Check for inconsistencies:** Are there inconsistencies between the source system and the target system?
- **Check for inaccurate data:** Is the data correct and does it reflect the actual entity being measured?
- **Check for duplicate data:** Does this data already exist within the target system?

# Sample Data Dictionary and Data Lineage

As a BI professional, using tools like data dictionaries and data lineages helps ensure data integrity and consistency from source to destination. Below are examples to illustrate how these tools function.

**Data Dictionary**

A **data dictionary** is a repository that provides details about the data objects in a database, including their content, format, and relationships. Here's an example using a product table from a sales database:

**Product Table Example**

| Item_ID | Price | Department | Number_of_Sales | Number_in_Stock | Seasonal |
|---|---|---|---|---|---|
| 47257 | $33.00 | Gardening | 744 | 598 | Yes |
| 39496 | $82.00 | Home Decor | 383 | 729 | Yes |

| Item_ID | Price | Department | Number_of_Sales | Number_in_Stock | Seasonal |
|---|---|---|---|---|---|
| 73302 | $56.00 | Furniture | 874 | 193 | No |
| 16507 | $100.00 | Home Office | 310 | 559 | Yes |
| 1232 | $125.00 | Party Supplies | 351 | 517 | No |
| 3412 | $45.00 | Gardening | 901 | 942 | No |
| 54228 | $60.00 | Party Supplies | 139 | 520 | No |
| 66415 | $38.00 | Home Decor | 615 | 433 | Yes |
| 78736 | $12.00 | Grocery | 739 | 648 | No |
| 34369 | $28.00 | Gardening | 555 | 389 | Yes |

**Data Dictionary Entries**

| Name | Definition | Data Type |
|---|---|---|
| Item_ID | ID number assigned to all product items in-store | Integer |
| Price | Current price of product item | Decimal |
| Department | Which department the product item belongs to | String |
| Number_of_Sales | Current number of product items sold | Integer |
| Number_in_Stock | Current number of product items in stock | Integer |
| Seasonal | Whether the product is seasonally available | Boolean |

**Usage**

The data dictionary allows for validation of incoming data. If incoming data has an error, such as numerical values in the Department field (which should be character-type), it can be flagged before ingestion.

---

**Data Lineages**

**Data lineage** refers to the tracking of data's origins, movements, and transformations within a system. This is essential for identifying issues and ensuring data quality.

**Example Scenario**

Imagine a scenario where an error is flagged regarding the number of sales for a specific product. To resolve this, you can trace the data's lineage through its lifecycle:

1. **Source Database**: The original data for sales numbers is recorded in an operational database.

2. **Transformation**: The data is processed through an ETL pipeline, where it might undergo transformations (e.g., cleaning, aggregating).
3. **Loading**: After processing, the data is loaded into the data warehouse.
4. **Reporting**: Finally, the data is used in BI reports and dashboards.

**Visual Representation**

```
[Operational Database] --> [ETL Process] --> [Data Warehouse] --> [BI Reports]
```

**Finding the Issue**

By examining the data lineage, you discover that the error originated from the operational database, where data entry inaccuracies occurred. This knowledge allows you to implement better validation checks at the source to prevent similar issues in the future.

# Business Rules in Database Management

Business rules are statements that impose constraints on specific parts of a database, reflecting how an organization utilizes its data. These rules not only enhance efficiency but also establish checks and balances that align with the organization's values and operational practices. For example, a company prioritizing cross-functional collaboration might require that data completion be confirmed by representatives from multiple teams.

**Imposing Business Rules**

The development and implementation of business rules are tailored to an organization's specific data needs. This variability underscores the importance of verifying that the database adheres to these rules.


**Verifying Business Rules**

After implementing business rules, it's crucial to continuously verify that they are functioning correctly. This involves checking that the data being imported adheres to the defined rules. For instance, the system should successfully route purchase order requests requiring approval to the appropriate stakeholders.

By integrating and verifying business rules within database systems, organizations can maintain data integrity and support informed decision-making.

# Database Performance Testing in an ETL Context

Database performance is crucial in the context of ETL (Extract, Transform, Load) processes because it directly affects the efficiency and reliability of data pipelines. A well-performing database ensures that data is extracted, transformed, and loaded quickly, enabling timely insights for decision-making.

**Key Factors Affecting Database Performance**

1. **Workload:** The volume of data and the complexity of queries impact how well a database can perform under pressure.
2. **Throughput:** The number of transactions processed in a given timeframe reflects the system's capacity.
3. **Resources:** Adequate CPU and memory are necessary for efficient processing of requests.
4. **Optimization:** Properly optimized queries and indexing can significantly improve response times.
5. **Contention:** Multiple users or processes trying to access the same resources can lead to bottlenecks.

**Essential Performance Testing Checks**

When conducting database performance testing in an ETL context, consider the following:

1. **Query Optimization:** Ensure that queries are structured efficiently to minimize processing time.
2. **Indexing:** Verify that the database is fully indexed to speed up data retrieval.
3. **Defragmentation:** Regularly defragment data to maintain optimal performance.
4. **Resource Availability:** Confirm that there are sufficient CPU and memory resources available.
5. **Row and Table Counts:** Cross-check the counts of tables, columns, and rows between source and target databases to ensure data integrity.

**Importance of Row and Table Count Testing**

Testing row and table counts helps to identify potential issues in the ETL process. Discrepancies in counts can indicate:

- **Data Loss:** Some records may not have been transferred correctly.
- **Duplicates:** Extra records might have been created inadvertently.
- **ETL Bugs:** Bugs in the ETL logic that need fixing to ensure accurate data migration.

**Query Execution Plan (QEP) Analysis**

Analysing the Query Execution Plan provides insights into how queries are executed by the database. This helps in:

- **Identifying Bottlenecks:** Understanding where delays occur in query processing.
- **Optimizing Performance:** Making adjustments based on how data is accessed and manipulated.

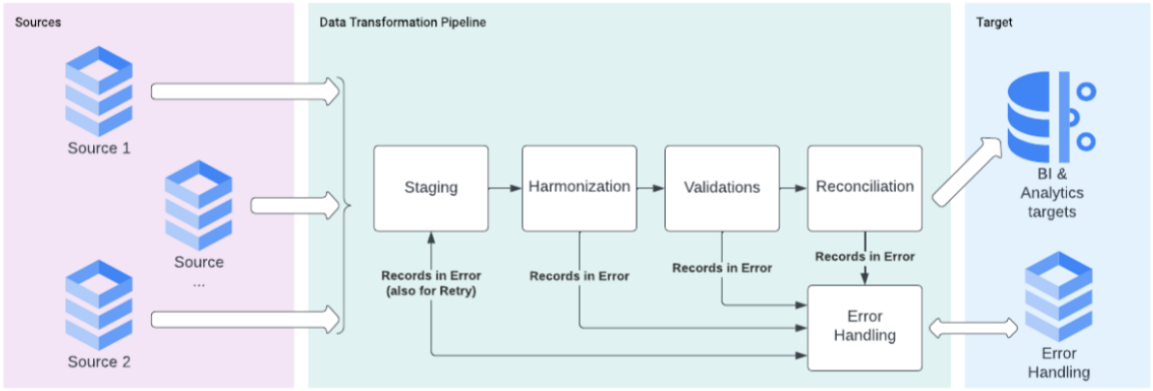
# Defending Against Known Issues in a Data Pipeline

Defensive checks are crucial for maintaining the integrity and accuracy of data throughout the ETL process. They help identify and mitigate issues before data is finalized and loaded into the target system. Here's how a Business Intelligence Analyst, effectively implemented these checks in data pipeline.

**Scenario Overview**

Each stage of pipeline was designed to ensure that data is correctly harmonized, validated, and reconciled.

**Pipeline Layers**




The pipeline consists of four key layers, each performing specific functions and incorporating defensive checks:

1. **Staging Layer**
    - **Purpose:** Raw data from source systems is stored here for initial processing.
    - **Defensive Checks:**
        - **Record Count Comparison:** Ensured the number of records received matched those stored.
        - **Row Integrity Checks:** Identified any extra or lost records by comparing rows.

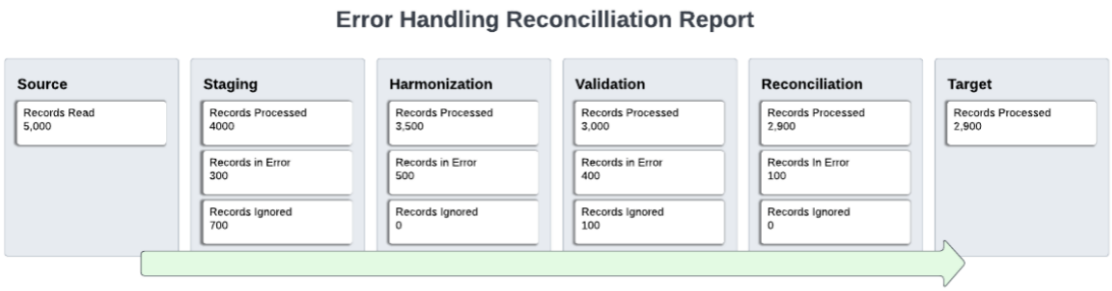
        - **Field Validation:** Important fields (amounts, dates, IDs) were verified for accuracy.
    - **Error Handling:** Mismatched records were logged, including details for tracking and resolution.

2. **Harmonization Layer**
    - **Purpose:** Normalizes and enriches data for consistency across sources.
    - **Defensive Checks:**
        - **Data Standardization:** Date formats, currencies, and case stylization were standardized.
        - **ID Formatting:** Ensured IDs were formatted correctly (e.g., leading zeros).
        - **Record Enrichment:** Applied conversion rules to enhance data quality.
    - **Error Handling:** Unharmonized records were sent to error handling for further analysis.

3. **Validations Layer**
    - **Purpose:** Applies business rules to validate data.
    - **Defensive Checks:**
        - **Null Value Checks:** Ensured crucial fields like "department" were not null.
        - **Authorized Value Checks:** Confirmed that "service type" values were within acceptable parameters.
        - **Contract Validity Checks:** Verified that billing records matched processed contracts.
    - **Error Handling:** Invalid records were recorded and moved to error handling.

4. **Reconciliation Layer**
    - **Purpose:** Detects duplicates and verifies record legitimacy.
    - **Defensive Checks:**
        - **Identification of Slow-Changing Dimensions:** Ensured accurate historical data tracking.
        - **Duplicate Checks:** Found and handled duplicate records appropriately.
        - **Aggregation Verification:** Ensured that aggregations were accurate and legitimate.
    - **Error Handling:** Records failing reconciliation checks were logged for review.

**Error Handling and Reporting**




After processing through all layers, compile an error handling report that summarized:

- The total number of records from each source.
- Counts of records marked as errors or ignored at each layer.
- Final counts of processed records that passed all checks.

This report not only provided insights into the pipeline's performance but also helped identify areas needing improvement, ensuring continuous monitoring and enhancement of the data pipeline.