About the Author

Dr. Hrishikesh (Hrish) Desai
CPA, CFA, EA, CMA, FCA

Associate Professor of Accounting

Director of the Master of Accountancy with Data Analytics Program

Arkansas State University

Contact Information

Linkedin | Twitter | Instagram

Data Storage, Processing, and Repositories

Understanding how organizations store, process, and manage data is fundamental to modern business operations. This comprehensive guide explores the technologies and methodologies that enable effective data management.

What is Data Storage?

Data storage refers to technology specifically designed to retain information and facilitate accessibility for authorized users. Effective storage systems enable organizations to perform business activities efficiently while maintaining data integrity and security.

Organizations utilize various storage types depending on their operational needs, data volume, and analysis requirements. Each storage type serves distinct purposes in the data lifecycle.

Operational Data Store (ODS)

An Operational Data Store (ODS) is a central database designed to integrate data from various operational systems in near real-time.

1

Integrated Data

Unifies data from disparate sources into a single view.

2

Near Real-time

Provides current operational data for immediate use.

3

Immediate Reporting

Supports instant analysis and decision-making.

4

Pre-Data Warehouse

Acts as an intermediary before data moves to a data warehouse.

Example: GadgetHub's ODS

GadgetHub, an e-commerce company, uses an ODS to unify transactional data.

- Integrates data from order, inventory, and customer service platforms.
- Provides a single, up-to-date picture of daily activities.
- Enables instant tracking of customer orders and inventory levels.

System Latency in ODS

System latency is the time delay between when data is generated in a source system and when it becomes available and usable in another system, such as an Operational Data Store (ODS). This delay can range from milliseconds to minutes.

For an ODS, high latency means stale data, which diminishes its primary purpose of providing a near real-time view of operations and impacts immediate decision-making.

Impact of Latency on Business Operations

E-commerce

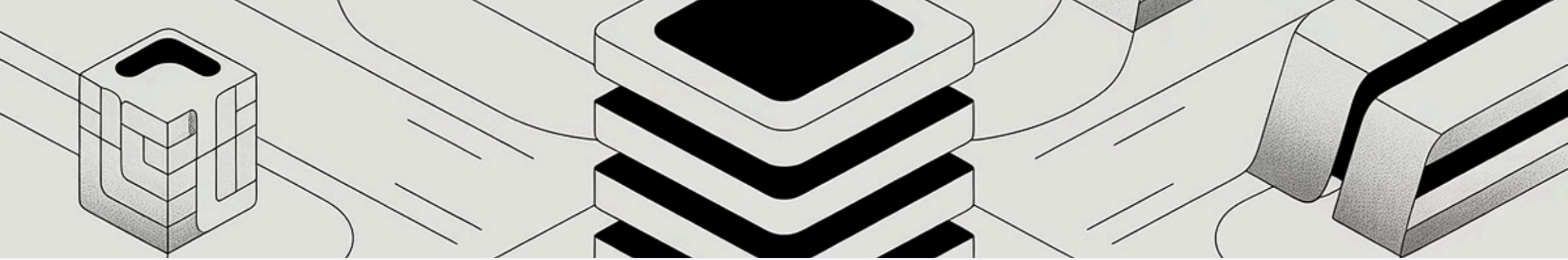
Delayed inventory updates can lead to overselling products and customer dissatisfaction.

Customer Service

Representatives give outdated information on orders or product availability, frustrating customers.

Financial Trading

Missing critical market opportunities due to delayed, non-real-time market data.



Data Warehouses

Purpose and Design

Data warehouses are centralized, large-scale repositories optimized for reporting and analysis rather than transactional processing. They consolidate data from multiple enterprise systems into a unified structure.

Unlike operational systems, warehouses are designed for complex queries and historical analysis, supporting strategic decision-making across the organization.

Data Flow

Information flows either directly from transactional systems or through an ODS before reaching the warehouse. This consolidated repository enables:

- Cross-functional reporting
- Data mart creation
- Advanced analytics
- Historical trend analysis

Understanding Data Lakes

A data lake represents a fundamentally different approach to data storage. Unlike structured warehouses, data lakes accommodate both structured and unstructured data in its natural, raw format.

No Predefined Schema

Data lakes don't require upfront structure definition. Information is stored in its original form without indexing or preprocessing.

Flexible Access

Users can access data in its raw state, applying structure and transformation only when needed for specific analysis.

Diverse Data Types

Accommodates documents, images, videos, sensor data, and traditional structured data in one repository.

Data Warehouse vs. Data Lake

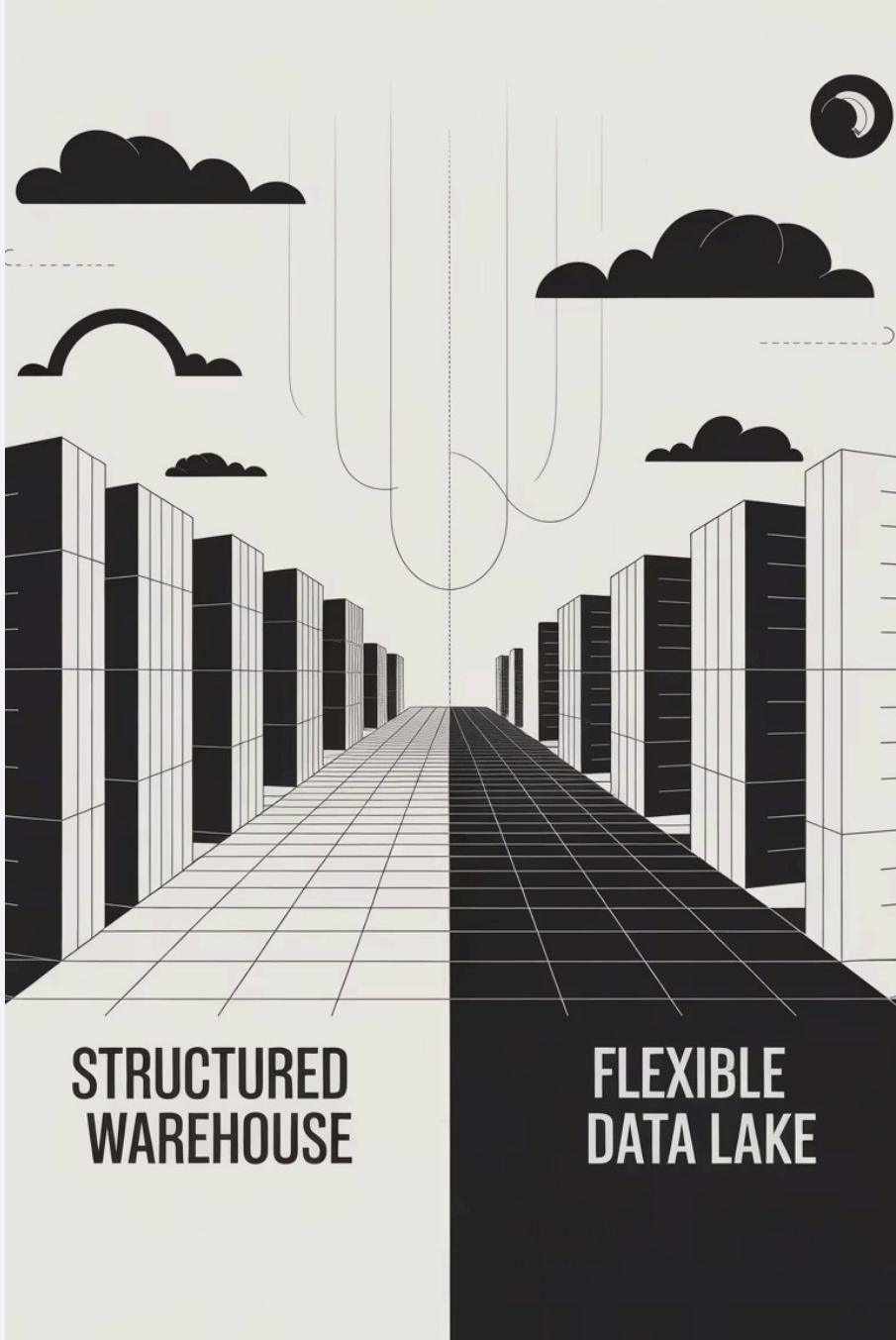
Data Warehouse

Structured, schema-on-write approach with predefined organization for rapid querying and analysis

Data Lake

Flexible, schema-on-read approach storing raw data without predetermined structure

The choice between these approaches depends on organizational needs, data types, and analytical requirements. Many organizations now implement both in complementary roles.



Schema-on-Write vs Schema-on-Read

In the context of data management, "schema" refers to the logical description of an entire database or data structure. It defines how the data is organized, including table names, column names, data types, and the relationships between different data elements. A schema acts as a blueprint for how data is structured and stored.

Schema-on-Write (Data Warehouses)

In a Schema-on-Write approach, the data schema is defined and enforced before any data is loaded into the storage system. This means that data must conform to a predefined structure upon ingestion. Any data that doesn't fit the schema is either rejected or requires transformation before it can be stored.

Schema-on-Read (Data Lakes)

With Schema-on-Read, data is loaded into the storage system in its raw, original format, without any predefined schema or structural enforcement. The schema is applied only when the data is read or queried, at the time of analysis. This allows for greater flexibility in handling diverse data types.

Schema-on-Write

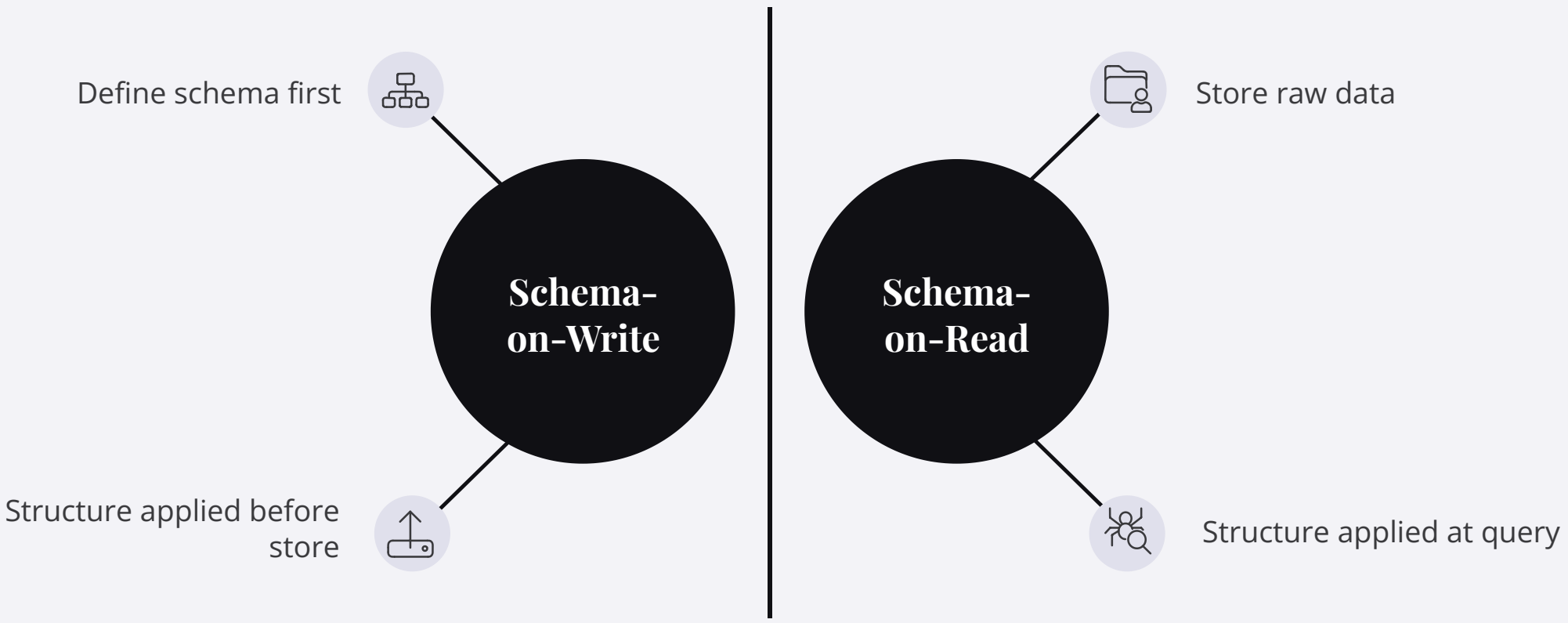
When to Use:

Ideal for traditional data warehouses, business intelligence, and reporting where data sources are well-understood and stable, and strict data governance is required.

Schema-on-Read

When to Use:

Best suited for data lakes, big data analytics, machine learning, and exploratory analysis where data sources are diverse, rapidly changing, or unknown at ingestion time.



Data Marts: Focused Insights

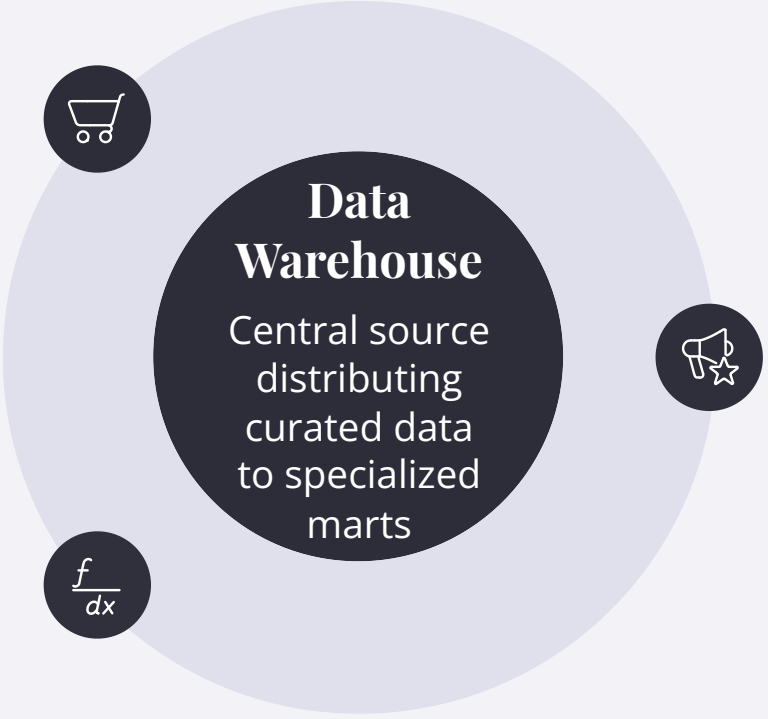
Data marts are specialized, subject-oriented subsets of a data warehouse, designed to serve the specific analytical needs of a particular business unit or department, such as sales, marketing, or finance.

Sales Data Mart

Revenue, Customers, Products

Finance Data Mart

Budgets, Expenses, Profits



Marketing Data Mart

Campaigns, Leads, Conversions

Key Advantages of Data Marts

Subject-Focused

Tailored to specific departmental needs, providing relevant data without unnecessary noise.

Enhanced Performance

Smaller data volumes lead to faster query execution and improved report generation times.

Departmental Autonomy

Empowers business units with greater control over their data and analytics.

Simpler Maintenance

Easier to manage, update, and secure due to their reduced scope and complexity.

Data marts are ideal for department-specific reporting and analytics, allowing teams to gain quicker, more relevant insights from their data for specialized decision-making.

Comprehensive Data Architecture

Understanding a modern data architecture requires visualizing how various systems integrate to manage, process, and analyze data from diverse sources, supporting different business needs from operational reporting to advanced analytics.

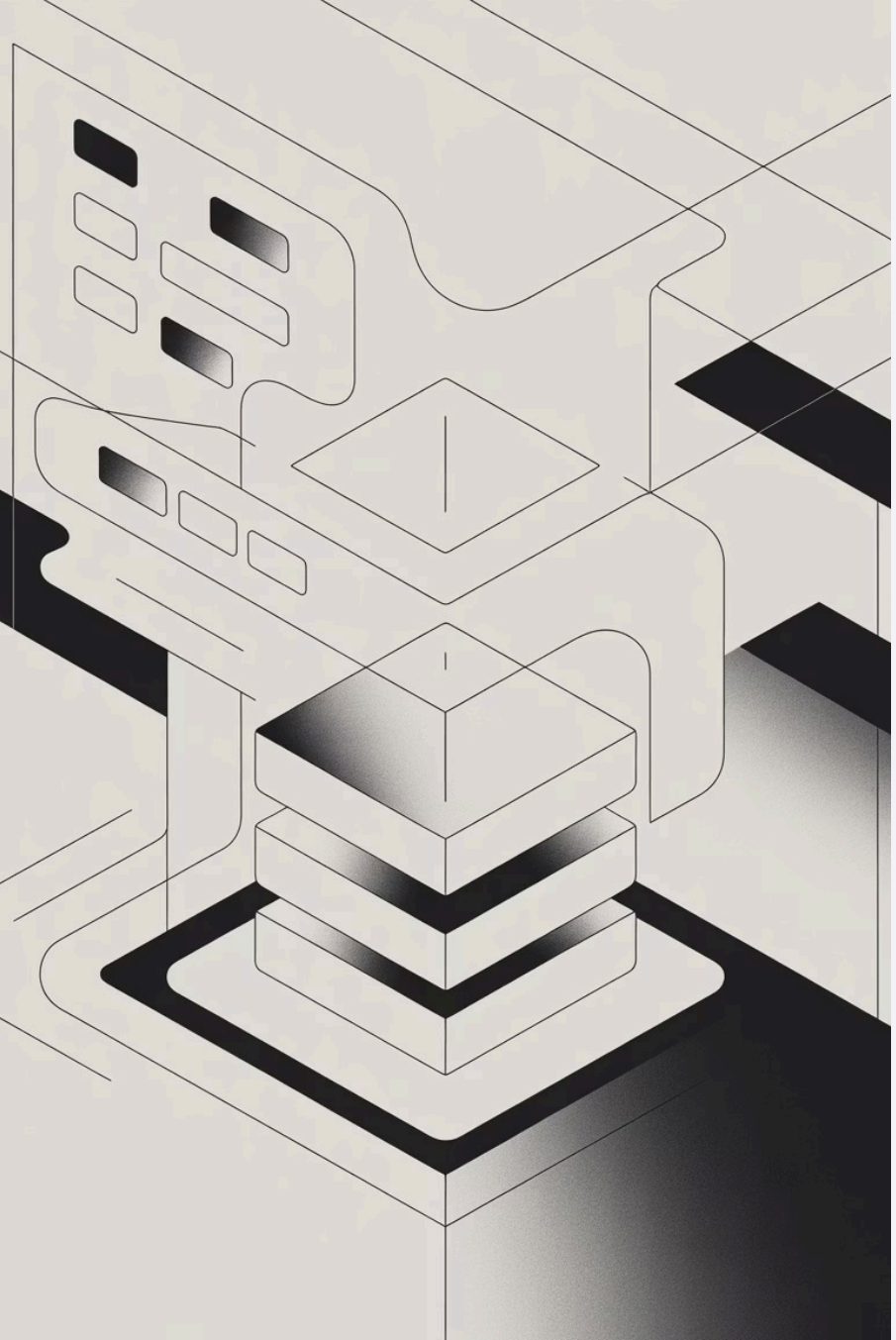
Data Sources: CRM, ERP, Web, IoT, Social Media

Operational Data Store: Current operational data for immediate reporting (Near Real-time Integration)

Data Storage Systems: Data Warehouse, Data Lake, Data Marts

ETL/ELT & Data Governance, Transformation, Cleansing, Validation, Data Quality

- **ETL = transform before load → cleaner, faster queries → structured data.**
- **ELT = load before transform → flexible, scalable → supports raw and varied data.**



Data Models vs. Database Schemas

While related, data models and database schemas serve different purposes in database design. Understanding this distinction is crucial for effective database development.

Data Model

High-level conceptual representation of data structures in an information system—the blueprint or design

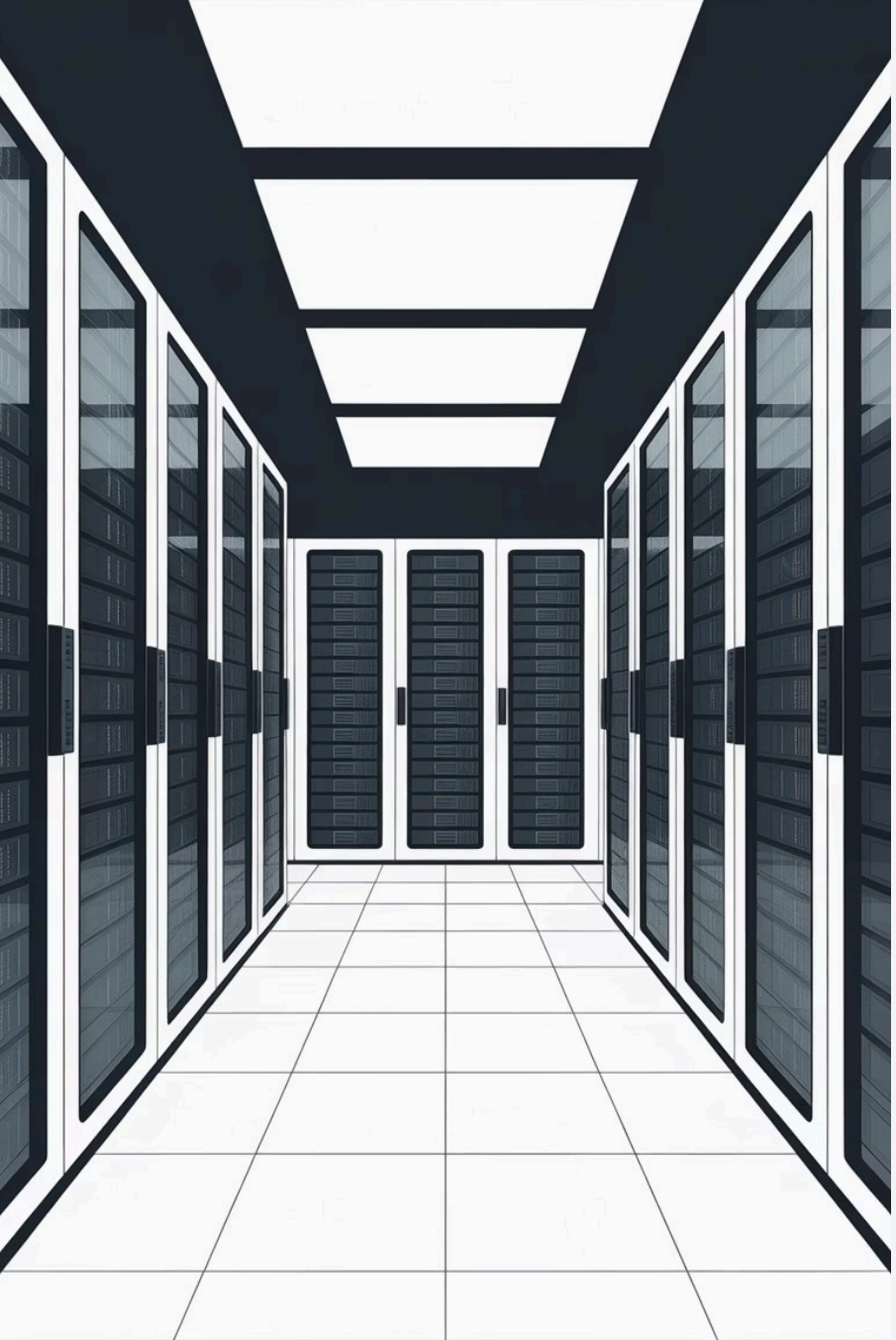
Database Schema

Actual implementation and execution of that design in a specific relational database—the construction

Understanding Database Schemas

A database schema provides instructions to the database engine on how to organize data in compliance with data models. It defines the actual structure including tables, columns, and relationships between data entities.

The schema specifies precisely how data will be stored and accessed in the database, translating conceptual designs into concrete technical specifications that the database management system can execute.



On-Premises Storage

Traditionally, organizations stored all data in physical drives and servers located within their own data centers. This approach, known as on-premises storage, provided direct control over infrastructure and data.

Traditional Approach

Companies maintained physical servers housing operational data stores, data warehouses, data marts, data lakes within their facilities.

Growing Challenges

As data volumes grew exponentially, maintaining on-premises infrastructure became increasingly costly and complex to manage effectively.

Evolution of Data Storage Technologies

Traditional Era

On-premises servers, flat files, early relational databases with limited scalability

1

2

Data Warehouse Era

Centralized repositories, dimensional modeling, business intelligence emergence

Big Data Era

Data lakes, cloud storage, handling structured and unstructured data at scale

3

4

Modern Era

Hybrid approaches, real-time analytics, AI-driven insights from diverse data sources

Cloud-Based Storage Revolution

Secure Migration

Data moves securely from on-premises to cloud infrastructure

Scalability

Organizations scale storage up or down based on current needs

Agility

Rapid data access and consumption across the enterprise

Cloud technologies have transformed data storage by offering highly scalable solutions that eliminate the burden of physical infrastructure maintenance while providing enhanced flexibility.

Introduction to Relational Databases

Among various data storage methods, relational databases stand out as one of the most efficient and effective approaches for structured data. These databases form the foundation of data warehouses and operational data stores.

Relational databases organize information into interconnected tables, providing a robust framework that ensures data quality, enforces business rules, and supports integrated business processes across the organization.

The Problem with Redundant Data

Storing the same information in multiple locations creates significant challenges for organizations. Normalized relational databases address these issues by maintaining a single version of truth.

Resource Waste

Duplicate data consumes unnecessary storage space, increasing infrastructure costs

Processing Overhead

Systems must reconcile multiple versions, requiring additional computational resources

Error Risk

Multiple copies increase the likelihood of data entry mistakes and inconsistencies

Relational Database Structure

Relational databases organize data across multiple related tables. Each table contains columns (attributes) and rows (records) filled with specific data values.

Core Components

- Tables store related data
- Columns define attributes
- Rows contain records
- Keys create relationships

Design Principles

Each column must be unique within its table and relevant to the table's purpose. Three types of columns exist: primary keys, foreign keys, and descriptive attributes.

Understanding Tables

Tables are organizational structures that establish columns and rows to store specific types of data records. In database design, tables are often called entities, representing objects within the system.

Customer Table Example

Stores customer names, addresses, phone numbers, and other relevant details

Employee Table Example

Contains employee information including names, departments, and hire dates

Inventory Table Example

Tracks product details, quantities, prices, and supplier information

Attributes: Describing Entities

Attributes, also known as columns, describe the characteristics or properties of each entity. They define what information the organization wants to know about each record.

> For example, in a Customer table, attributes might include "First Name," "Last Name," "Email Address," and "Phone Number." Each attribute must be unique within its table and relevant to the table's purpose.

The three types of attributes—primary keys, foreign keys, and descriptive attributes—work together to create a comprehensive data structure that supports business operations.

Records and Fields

Records (Rows)

Each row in a table represents one complete record containing information about a single entity. For instance, one row in a Customer table provides all stored information about one specific customer.

Fields (Cells)

A field exists at the intersection of a column and row. This space holds a specific data value—the actual information stored for that attribute of that particular record.

Data Types Matter

Every attribute in a table must have a designated data type that specifies how information is stored and analyzed. Choosing the correct data type is crucial for accurate analysis and reporting.

Numerical

For values requiring mathematical operations like sums or averages (e.g., price, quantity)

Text

For descriptive information that won't be used in calculations (e.g., names, addresses)

Date/Time

For temporal data enabling time-based analysis and calculations

Attributes with meaningful numerical values must be classified as numerical data types to ensure they function properly in mathematical equations. Misclassifying data types can lead to analysis errors.

Database Keys: Creating Unique Identity

While each attribute in a table must have a unique name, rows aren't inherently unique without database keys. Keys serve two critical functions: uniquely identifying each record and facilitating relationships between tables.

Understanding primary and foreign keys is essential for grasping how relational databases maintain data integrity and create meaningful connections between related information.

Primary Keys: Unique Identifiers

1

Required in Every Table

Each table must have a primary key to ensure every record is uniquely identifiable

2

Typically One Column

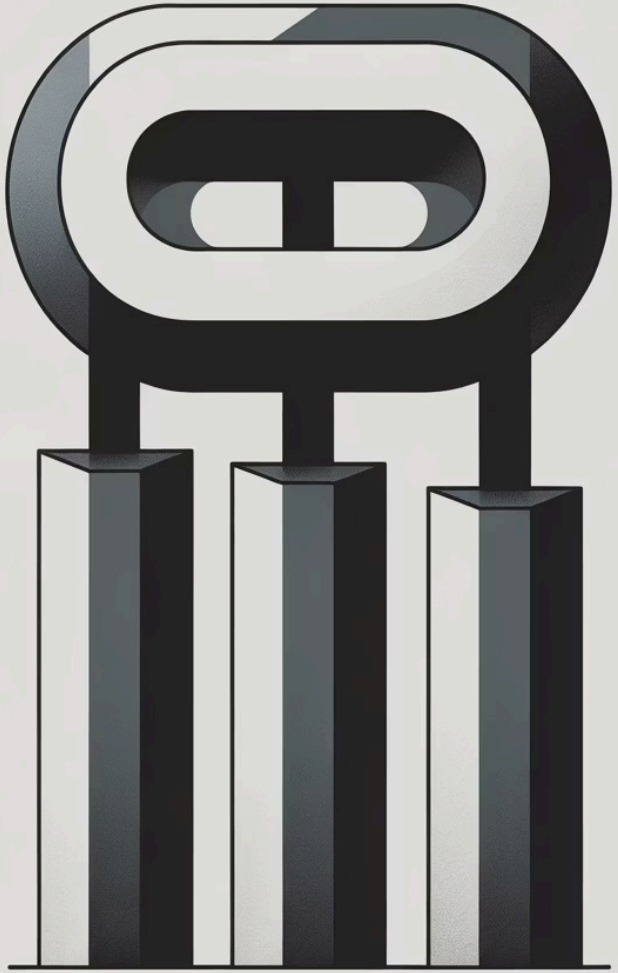
Usually consists of a single attribute, though composite keys use multiple columns

3

Not Descriptive

Often uses sequential numbers or letter combinations rather than meaningful descriptions

Common examples include student ID numbers, order numbers, invoice numbers, account numbers, and social security numbers. These identifiers ensure each record can be precisely referenced.



Composite Primary Keys

Sometimes a single attribute cannot uniquely identify each record in a table. In these cases, multiple attributes combine to form a composite primary key.

> Consider a course enrollment system: neither Student ID alone nor Course ID alone uniquely identifies an enrollment record, since students take multiple courses and courses have multiple students. Together, however, Student ID and Course ID create a unique combination for each enrollment.

Foreign Keys: Building Relationships

Foreign keys are attributes that appear as primary keys in one table but exist in another table to create relationships. This mechanism is fundamental to how relational databases connect related information.

1

Primary in One Table

Customer ID serves as the primary key in the Customer table

2

Foreign in Another

The same Customer ID appears as a foreign key in the Orders table

3

Creates Relationship

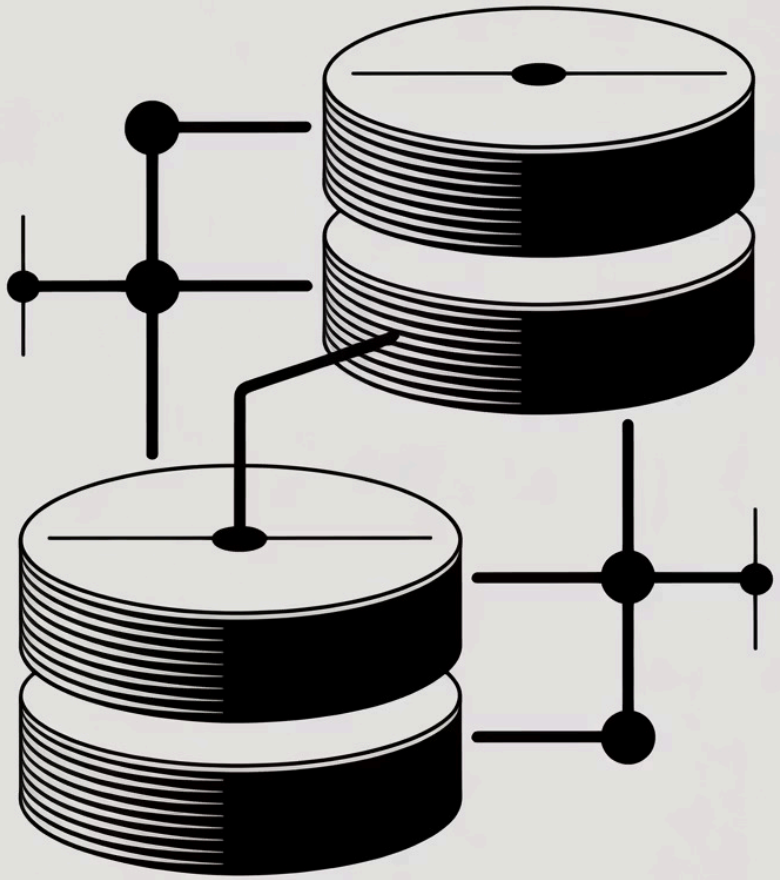
This link connects customer information to their orders

Foreign keys can appear multiple times in a table—one customer may place many orders—but each reference points back to a single unique record in the related table.

How Relationships Work

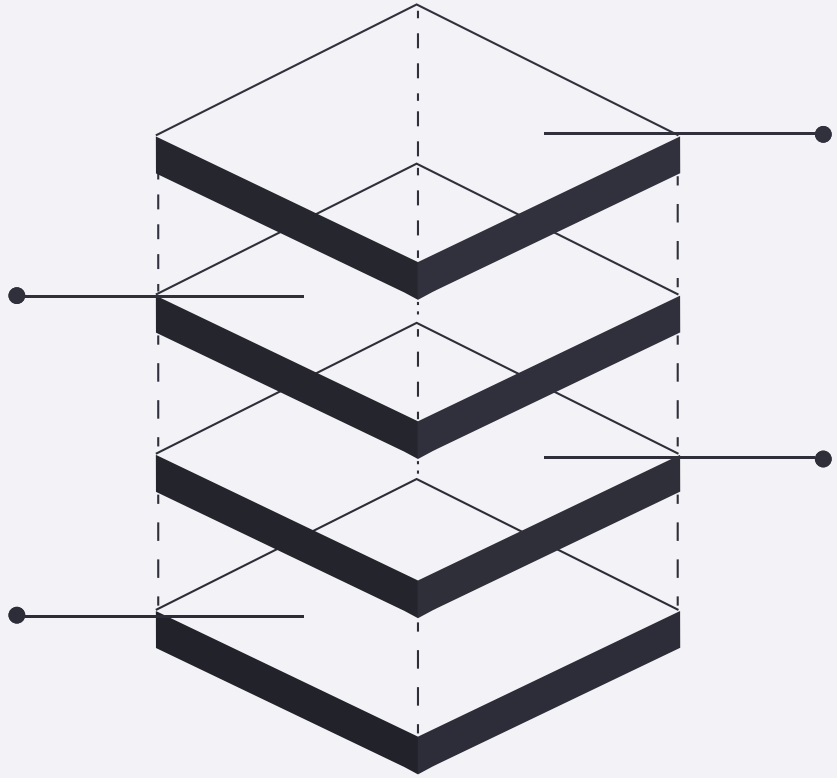
The connection between a primary key in one table and a foreign key in another creates the relationship between tables. Whenever two tables need to be related, one must contain a foreign key referencing the other's primary key.

This design ensures data integrity while avoiding redundancy. Instead of storing complete customer information with every order, the system stores it once in the Customer table and references it through the foreign key.



E-commerce Database Schema in Action

This diagram illustrates a practical e-commerce database schema, demonstrating how primary keys (PK), foreign keys (FK), and descriptive attributes work together to organize and relate critical business information across multiple tables.



Orders

Order ID (PK), Customer ID (FK), Order Date, Total Amount

Customers

Customer ID (PK), First Name, Last Name, Email, Address

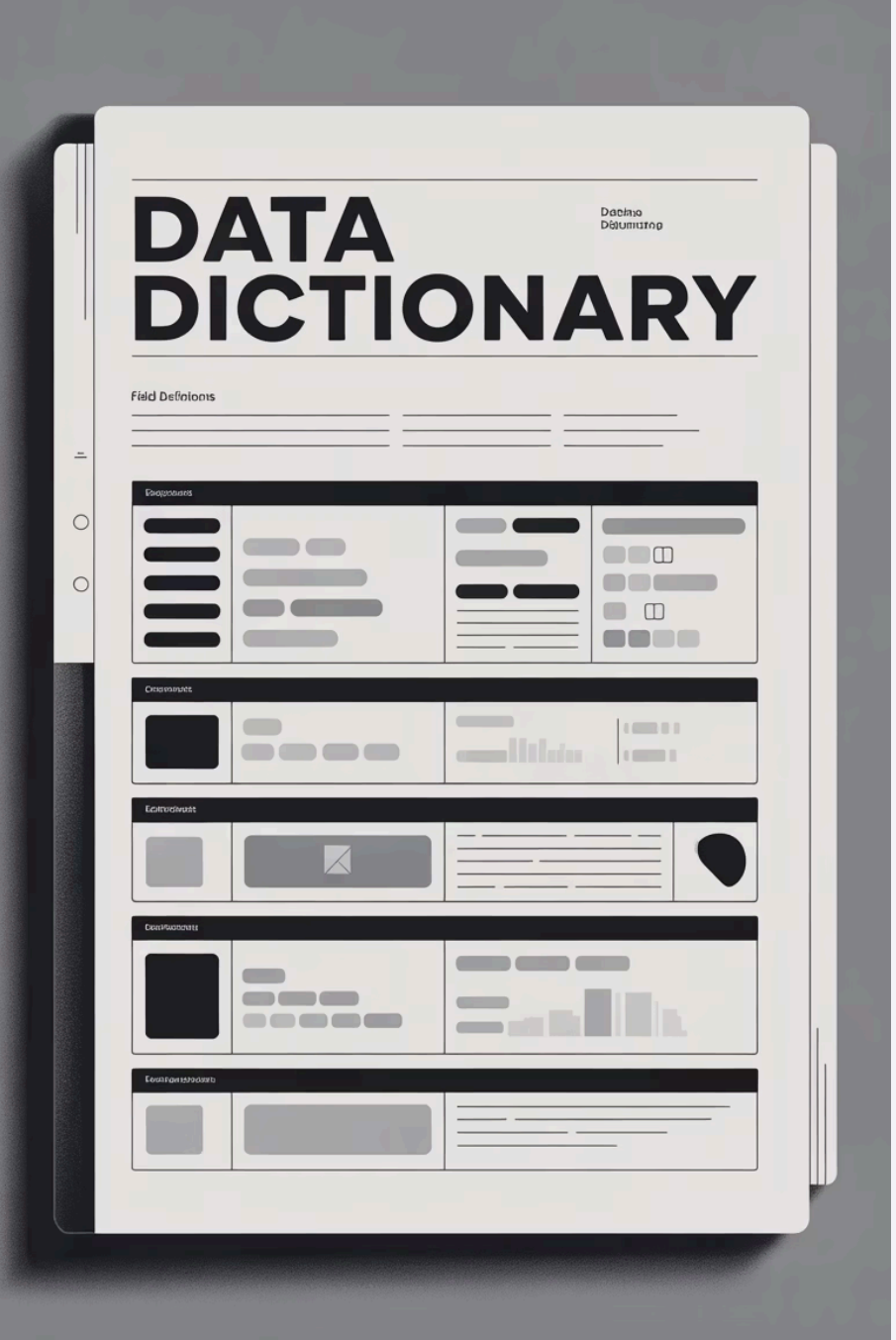
Order Details

Order Detail ID (PK), Order ID (FK), Product ID (FK), Quantity, Unit Price

Products

Product ID (PK), ProductName, Price, StockQuantity

Observe how foreign keys link records between tables, enabling the system to understand which customer placed which order, and what products were included in each order, all while maintaining data integrity.



Data Dictionaries: Metadata Explained

Data dictionaries are a form of metadata—data about data. They summarize information about database contents, making it easier to work with data and understand how it informs decisions and reports.

Basic Information

At minimum, data dictionaries detail attribute names and descriptions, clarifying what each field represents and how it should be used.

Preventing Mistakes

Descriptions ensure data is analyzed appropriately. Without clear definitions, users might misinterpret fields, leading to incorrect analysis and flawed decisions.

Comprehensive Data Dictionary Contents

For relational databases, data dictionaries track extensive attribute information beyond simple names and descriptions:

- **Key Type**

  Whether the attribute is a primary key, foreign key, or descriptive attribute

- **Required Status**

  Whether the field must contain a value or can be left empty

- **Data Type**

  The format of data stored (numerical, text, date/time, etc.)

- **Default Values**

  Any automatic values assigned when no input is provided

- **Field Size**

  Maximum character length or numerical range permitted

Data Dictionary Example: Customer Table

This table illustrates a typical data dictionary entry for a **Customer** table, detailing each attribute's characteristics. This documentation is vital for database administrators and developers to ensure data integrity, consistency, and proper usage across systems.

Field Name	Data Type	Description	Key Type	Required	Length/Constraints
CustomerID	INT	Unique identifier for each customer	Primary Key	Yes	Auto-incrementing integer
FirstName	VARCHAR	Customer's given name	Descriptive	Yes	Max 50 characters
LastName	VARCHAR	Customer's family name	Descriptive	Yes	Max 50 characters
Email	VARCHAR	Customer's primary email address	Descriptive	Yes	Max 100 characters, UNIQUE
PhoneNumber	VARCHAR	Customer's primary phone number	Descriptive	No	Max 20 characters
Address	VARCHAR	Full mailing address of the customer	Descriptive	No	Max 255 characters
RegistrationDate	DATETIME	Date and time of customer account creation	Descriptive	Yes	Default: Current Timestamp

Database Normalization: Eliminating Redundancy

Normalization is a database design technique that reduces data redundancy and eliminates problematic characteristics like insertion, update, and deletion anomalies.

The normalization process divides larger tables into smaller, related tables linked through relationships. This approach eliminates repetitive data and ensures information is stored logically.

First Normal Form (1NF)

The first step in normalization establishes foundational rules that make sorting and filtering data easier. First normal form has two essential criteria:

Atomic Values

Each cell must contain only one piece of information—no multiple values in a single column

Unique Identification

Every record must be uniquely identified through a primary key (single or composite)

> Example violation: A "Phone Numbers" field containing "555-1234, 555-5678" violates 1NF because it stores multiple values. Each phone number should be a separate record or field.

Second Normal Form (2NF)

Once a table achieves 1NF, the next step ensures all non-key attributes depend on the entire primary key. This rule particularly matters for tables with composite primary keys.

Meets 1NF Requirements

Table must first satisfy all first normal form criteria

Full Dependency

Every non-key attribute must describe the complete primary key, not just part of it

Consider an Order Details table with composite primary key (Order ID, Product ID). An attribute like "Quantity Ordered" depends on both keys—it describes how many of a specific product were ordered in a particular order. However, "Product Name" only depends on Product ID, violating 2NF.

Third Normal Form (3NF)

Third normal form ensures each column describes only the primary key, eliminating transitive dependencies where non-key attributes depend on other non-key attributes.

The Rule

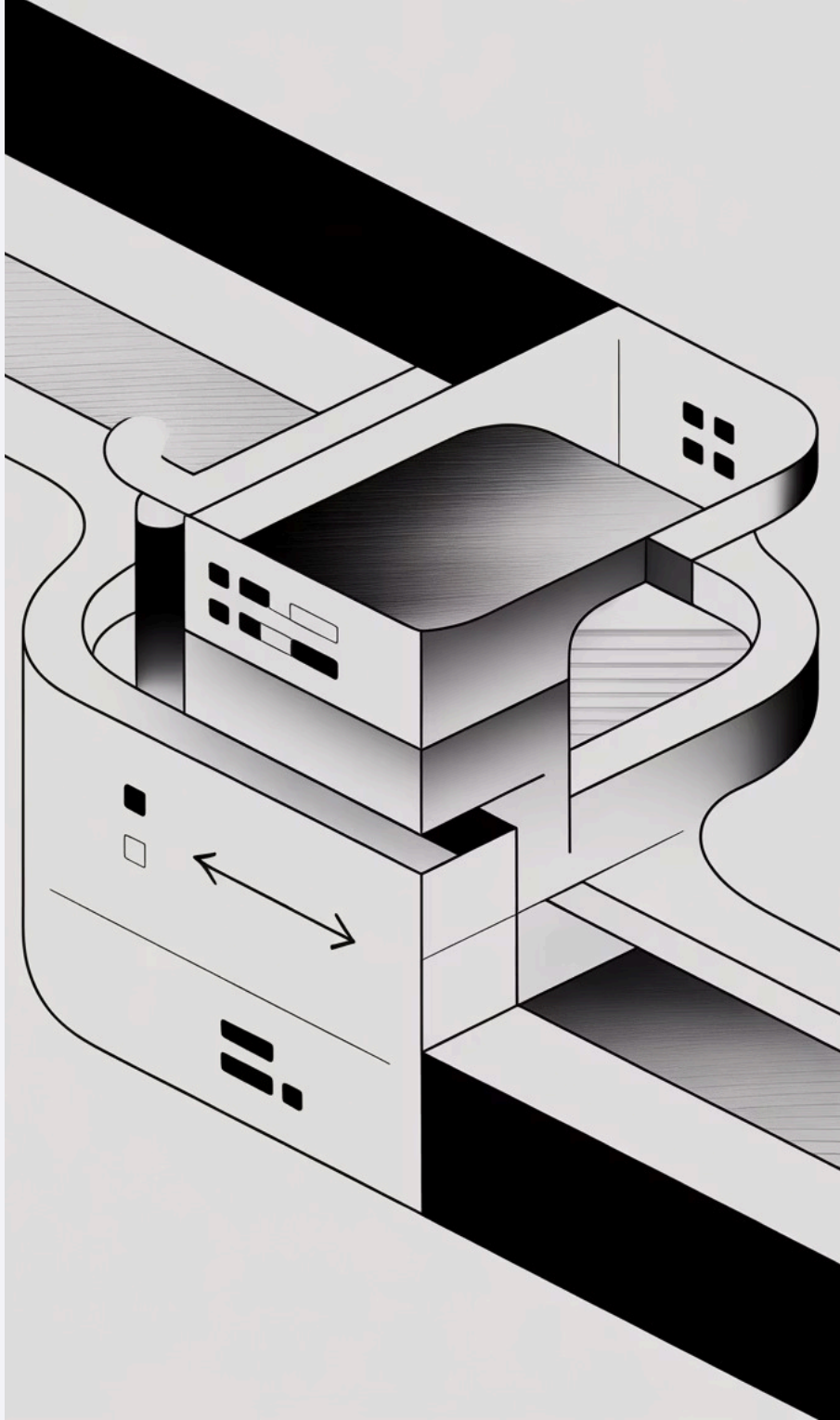
Non-key attributes must depend directly on the primary key, and not indirectly through other non-key attributes. This eliminates transitive dependencies, where a non-key attribute relies on another non-key attribute.

The Example

Consider an 'Employee' table with Employee ID (primary key), Employee Name, Department ID, Department Name, and Department Manager. Here, 'Department Name' and 'Department Manager' depend on 'Department ID' (a non-key attribute), not directly on 'Employee ID'. This is a transitive dependency and violates 3NF.

To resolve this, split into two tables:

- **Employee**: Employee ID, Employee Name, Department ID
- **Department**: Department ID, Department Name, Department Manager



Primary Key, Full Key, Only the Key

Remember the Normalization Mantra

1

Primary Key

Every table must have a primary key (1NF)

2

Full Key

Attributes depend on the entire composite key (2NF)

3

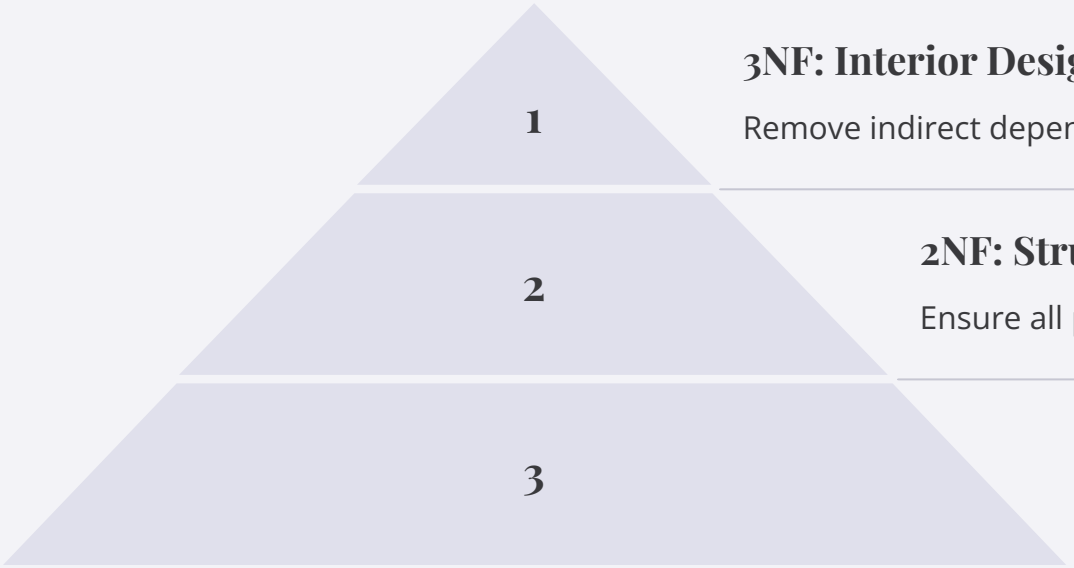
Only the Key

Attributes depend only on the primary key (3NF)

These forms are progressive: achieving 3NF requires first satisfying 2NF, which requires first satisfying 1NF. Each level builds upon the previous foundation.

Normalization Memory Aid

To truly grasp database normalization, let's build an analogy. Think of your database as a house, where each normal form represents a critical stage in its construction, ensuring a sturdy, well-organized, and efficient home for your data.



3NF: Interior Design

Remove indirect dependencies, making each room perfectly functional.

2NF: Structural Integrity

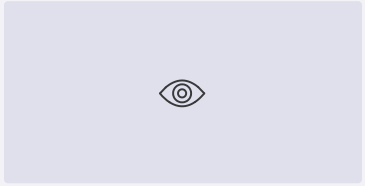
Ensure all parts of the house are directly supported by its main beams.

1NF: Solid Foundation

Each brick is unique and placed correctly, forming a stable base.

This house-building analogy helps visualize how each normal form builds upon the last, progressively refining your database structure for optimal integrity and efficiency.

Data Models



Conceptual

High-level overview for stakeholders

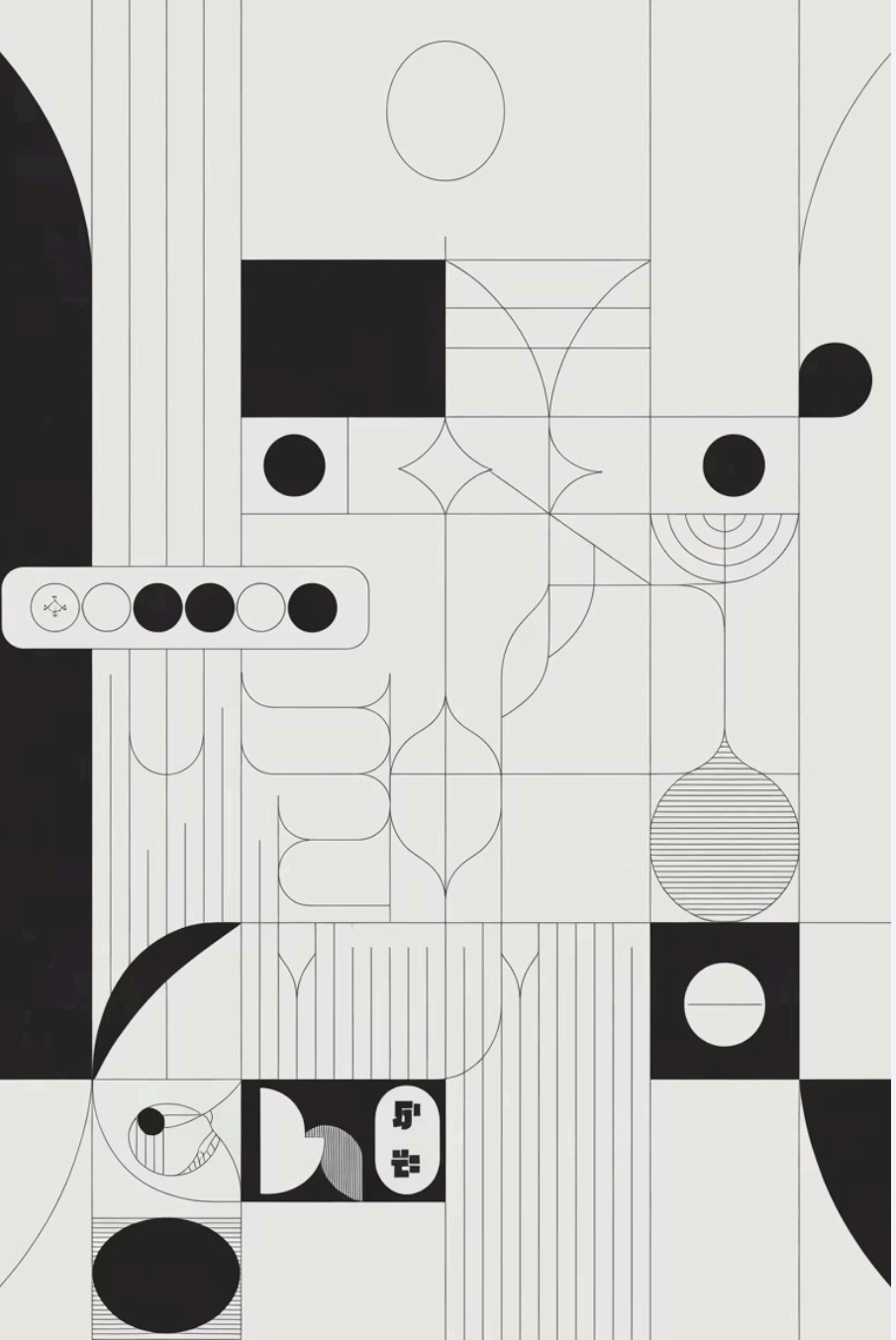
Logical

Detailed design with attributes and keys

Physical

Complete implementation specifications

Data models are created in stages, starting with conceptual (least complex), progressing through logical, and culminating in physical (most complex). Each level serves different audiences and purposes.



Conceptual Data Models

Conceptual data models provide a high-level, big-picture representation of data structures in an information system. They define main entities and relationships without delving into detailed attributes or physical implementation.

Purpose

A conceptual model for a sales system might show five entities: Customers, Employees, Orders, Products, and Departments. The model illustrates that Orders relate to Customers, Employees, and Products, while Products relate to Departments.

Simplicity

This high-level overview helps stakeholders understand the system's scope without getting lost in technical details. Once approved, additional details can be added through logical and physical models.

Logical Data Models

Logical data models provide more detailed representations than conceptual models, defining not only entities and relationships but also the attributes of each entity.

Includes Attributes

Shows all fields that will be stored for each entity

Identifies Keys

Designates primary and foreign keys for each table

Addresses Normalization

Resolves first and second normal form issues

Logical models are useful for data-oriented projects like designing data warehouses or developing new systems, providing sufficient detail for technical teams while remaining technology-independent.

Physical Data Models

Physical data models represent the most detailed level, specifying exactly how data will be stored in the database. These models are complete enough that databases can be built directly from their specifications.

Tables and Columns

Entities become tables, attributes become columns with specific names

Data Types

Each column has designated data type and size constraints

Implementation Details

Includes required fields, default values, and constraints

Common Data Types in Physical Models

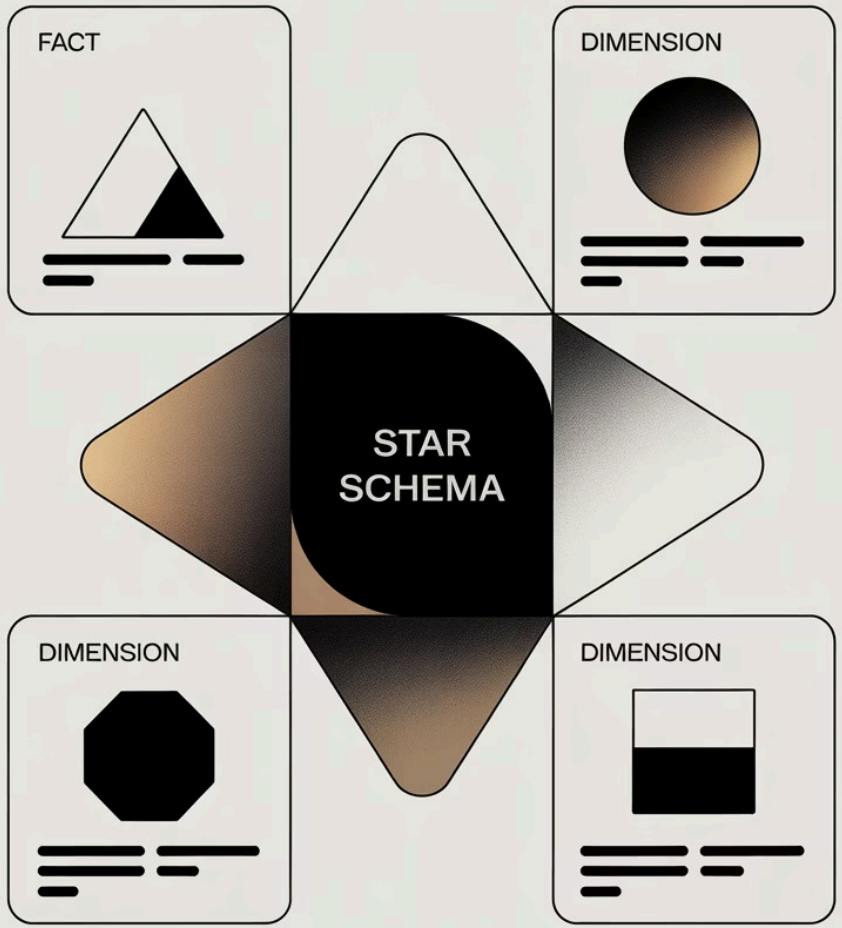
Data Type	Description and Usage
Integer (Int)	Whole numbers for mathematical calculations or identifiers; no decimal places
Character (Char)	Fixed-length text or number strings; character limit specified in parentheses
Date/Time	Temporal values enabling calendar-based calculations and time analysis
Decimal	Numbers with decimal places; supports precise mathematical operations
Text/String	Variable-length non-numeric values; includes letters, symbols, or numbers without mathematical meaning

Choosing appropriate data types ensures accurate storage, efficient processing, and correct analysis of information.

Comparing Model Complexity

Feature	Conceptual	Logical	Physical
Entity/Table Names	✓ (high-level entity names)	✓ (detailed table names)	✓ (actual table names)
Relationships	✓ (basic relationships shown)	✓ (detailed relationships)	✓ (implemented relationships)
Attributes/Columns	✗ (no detailed attributes)	✓ (all attributes defined)	✓ (actual column names)
Primary/Foreign Keys	✗ (keys not specified)	✓ (keys clearly identified)	✓ (implemented keys)
Data Types	✗ (no data types specified)	✓ (logical data types)	✓ (specific database data types like VARCHAR(50), INT, etc.)
Constraints	✗ (no constraints)	✓ (business rules and constraints)	✓ (all constraints implemented)
Database-Specific Details	✗ (platform independent)	✗ (still platform independent)	✓ (tailor for a specific DBMS)

Each model level builds upon the previous, adding progressively more detail. An effective logical model must include all conceptual elements, and an effective physical model must include all logical elements.



Dimensional Modeling: Star and Snowflake Schemas

While transactional databases use normalized designs (1NF, 2NF, 3NF), data warehouses and data marts often employ dimensional modeling with star or snowflake schemas for optimized reporting and analysis.

These schemas organize data into two table types: **fact tables** containing measurable business metrics, and **dimension tables** providing descriptive context for those measurements.

Fact Tables and Dimension Tables

Fact Tables

Contain quantitative measures or metrics that measure business performance—sales revenue, costs, profits, quantities. Include foreign keys linking to dimension tables for context.

Dimension Tables

Contain descriptive or contextual data that provides meaning to facts—dates, product names, customer information, locations. Attributes describe various aspects of each dimension.

For example, a sales fact table might contain revenue amounts (facts) with foreign keys to date, product, customer, and store dimensions (context).

Star Schema Design

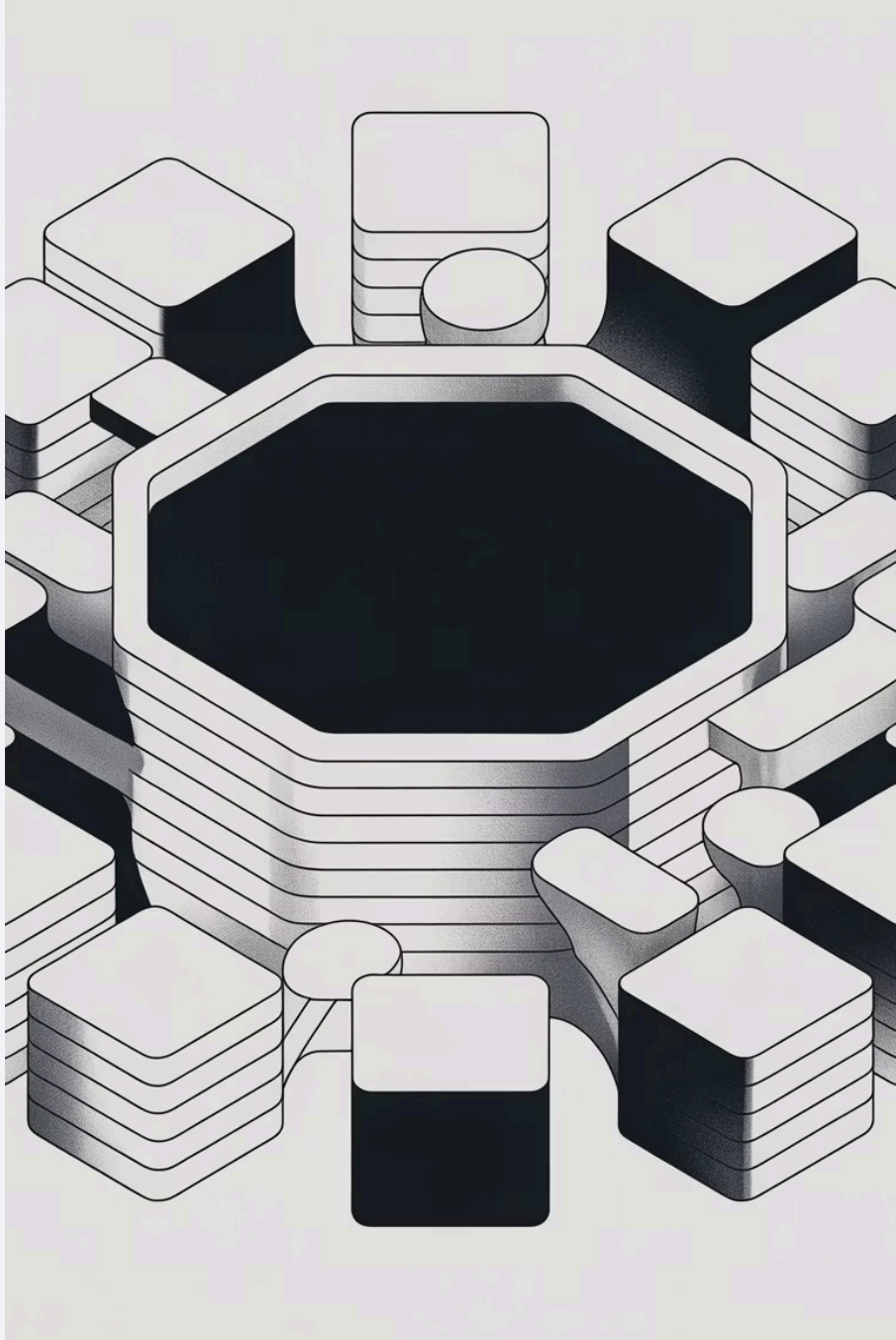
The star schema is the most common and simplest dimensional modeling approach. Data organizes around a central fact table with dimension tables radiating outward like points of a star.

Structure

One central fact table connects directly to multiple dimension tables. Each dimension table links to the fact table through a foreign key relationship.

Benefits

Simple structure makes queries intuitive and fast. Business users easily understand the model, facilitating self-service reporting and analysis.



Snowflake Schema Design

Snowflake schemas extend star schemas by further normalizing dimension tables. Dimensions break down into multiple related tables rather than single denormalized tables.

1

More Complex Structure

Dimension tables split into additional related tables, creating more foreign key relationships

2

Increased Flexibility

Allows storage of more detailed dimensional information with less redundancy

3

Balanced Approach

Strikes middle ground between fully normalized and star schema designs

Advantages of Dimensional Modeling

User-Friendly

Business users easily understand dimensional models, making them more accessible than highly normalized transactional databases for reporting purposes.

BI Tool Compatibility

Business intelligence tools like Tableau and Power BI work exceptionally well with star and snowflake schemas, enabling powerful visualizations.

Query Performance

Denormalized structure optimizes query performance for analytical workloads, providing faster insights from large datasets.

Dimensional vs. Normalized: Sales Analysis

For a retail company analyzing sales performance, the choice of data model significantly impacts query speed and ease of use.

Concrete Business Scenario: Sales Analysis

Business Question:

"What were our total sales by product category for each quarter last year?"

Normalized Database Approach:

- To answer this question, you'd need to JOIN multiple tables:
  - Orders table → OrderDetails table → Products table → Categories table → Customers table → Time/Date calculations
- Complex SQL with 4-5 table JOINs
- Slow performance due to multiple relationships
- Difficult for business users to write or understand

Dimensional Model (Star Schema) Approach:

- Simple structure: Sales Fact table in center with dimensions around it
- Sales Fact table contains: SalesAmount, Quantity, ProductKey, TimeKey, CustomerKey
- Dimension tables: Product (with Category), Time (with Quarter/Year), Customer
- Simple query: Just JOIN Sales Fact to Product and Time dimensions
- Fast performance, easy to understand

The Result:

- Normalized: Complex 5-table JOIN
- Dimensional: Simple 3-table JOIN

Trade-offs of Dimensional Modeling

While dimensional modeling offers significant advantages for analytics, it comes with important considerations regarding data management and maintenance.

Data Redundancy

Denormalized designs intentionally duplicate data, sacrificing the "one version of truth" principle of normalized databases.

Update Complexity

When changes are necessary, updates must occur in every location where data has been duplicated, increasing maintenance burden.

Storage Requirements

Redundant data consumes more storage space compared to fully normalized designs, though modern storage costs make this less concerning.

Choosing the Right Approach

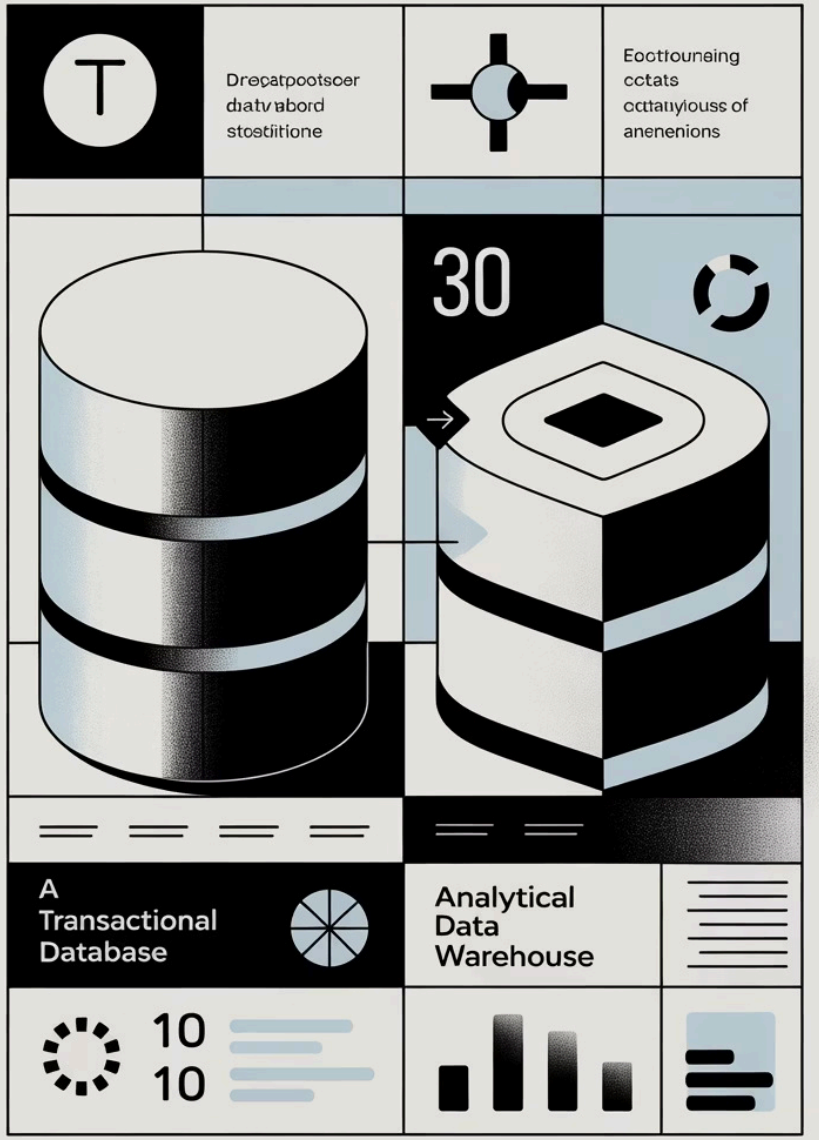
The choice between normalized transactional databases and dimensional data warehouses depends on the primary use case and business requirements.

Transactional Systems

Use normalized designs (3NF) for operational systems requiring data integrity, minimal redundancy, and frequent updates

Analytical Systems

Use dimensional models (star/snowflake) for reporting and analysis systems prioritizing query performance and user accessibility



The Importance of Data Governance

Effective database design extends beyond technical structure to encompass data governance—the policies, procedures, and standards that ensure data quality, security, and appropriate use.

Quality Standards

Establish rules for data accuracy, completeness, and consistency across systems

Access Controls

Define who can view, modify, or delete different types of data

Change Management

Document and control modifications to database structures and schemas