# CSCI E-103: Data Engineering for Analytics
# Lecture 02: Data Modeling and Metadata

Harvard Extension School

Fall 2024

- **Course:** CSCI E-103: Data Engineering for Analytics
- **Lecture:** Lecture 02: Data Modeling & Metadata
- **Instructor:** Anindita Mahapatra & Eric Gieseke
- **Objective:** Master data modeling techniques (conceptual, logical, physical), understand OLTP vs OLAP systems, compare DWH architectures, and learn data formats and compression

**Key Summary**

This lecture covers the core discipline of **data modeling**—organizing data to meet business needs. We explore the three modeling levels (conceptual, logical, physical), contrast OLTP and OLAP systems, examine dimensional modeling with Star and Snowflake schemas, and compare data warehouse architectures (Inmon, Kimball, Data Vault). We also cover metadata management, data formats (CSV, JSON, Parquet, Delta Lake), compression techniques, and data profiling. The lab introduces housing price prediction using linear regression.

## Contents

# 1 Review: Key Concepts from Lecture 01

Before diving into data modeling, let's reinforce the foundational concepts:

**Table 1:** *Lecture 01 Key Terms Review*

| Term | Key Explanation |
|---|---|
| **ACID** | Atomicity, Consistency, Isolation, Durability (relational databases, bank transactions) |
| **BASE** | Basically Available, Soft State, Eventual Consistency (NoSQL, social media) |
| **CAP Theorem** | Distributed systems can only have 2 of 3: Consistency, Availability, Partition Tolerance |
| **IaaS/PaaS/SaaS** | Cloud service models (Infrastructure, Platform, Software as a Service) |
| **Lakehouse** | Combines data lake (flexibility) + data warehouse (reliability) |
| **DAG** | Directed Acyclic Graph—defines workflow in data pipelines |
| **ETL** | Extract, Transform, Load—moving data between systems |
| **5 V's of Big Data** | Volume, Velocity, Variety, Veracity (accuracy), Value |
| **Spark Advantage** | In-memory processing (100x faster than Hadoop's disk-based MapReduce) |

**Warning**

**The Biggest Challenge: Data Quality and Staleness**

According to industry research, the biggest problems in big data aren't volume or velocity—they're ensuring data is **accurate** and **up-to-date**. "Garbage In, Garbage Out" (GIGO) remains the cardinal rule: no amount of sophisticated modeling can fix fundamentally bad data.

# 2 What is Data Modeling?

**Definition:**

Data Modeling Data modeling is the process of **organizing and structuring data** to meet business process requirements. It creates a "blueprint" that defines how data is stored, accessed, and related—transforming real-world complexity into a computer-understandable structure.

## 2.1 Why Data Modeling Matters

1. **Consistency and Quality**: Standardizes names, rules, and formats across the organization
2. **Efficient Storage and Retrieval**: Optimizes how data is persisted and queried
3. **Communication Tool**: Creates common vocabulary between business and technical teams
4. **Early Error Detection**: Catches inconsistencies in design phase, not production
5. **Documentation**: Serves as living documentation of data assets

**Key Information**

**Cost of Late Discovery**
Fixing a data model error during design costs $1. Fixing the same error in development costs $10. Fixing it in production costs $100+. Data modeling is an investment that pays dividends throughout the system's lifecycle.

## 2.2 The Three Levels of Data Modeling

Data modeling progresses from abstract business concepts to concrete technical implementations:

### 2.2.1 1. Conceptual (Semantic) Model

- **Purpose**: Define core business concepts and rules
- **Audience**: Business stakeholders, domain experts
- **Focus**: Entities and relationships (e.g., "Customer purchases Product")
- **Technical details**: None—purely business-focused

### 2.2.2 2. Logical Model

- **Purpose**: Define data structure, attributes, and relationships in detail
- **Audience**: Developers, data architects, business analysts
- **Focus**: Entity attributes, data types, keys, cardinality
- **Technical details**: Technology-agnostic (not tied to specific database)

### 2.2.3 3. Physical Model

- **Purpose**: Translate logical model into specific database technology

- **Audience**: DBAs, data engineers
- **Focus**: Table names, column types, indexes, partitions, constraints
- **Technical details**: Fully specified for target platform (e.g., PostgreSQL, Delta Lake)

**Example:**

Online Bookstore Example **1. Conceptual Model**: "Customer orders Book"

**2. Logical Model**:

- Customer(CustomerID [PK], Name, Email)
- Book(BookID [PK], Title, Author)
- Order(OrderID [PK], OrderDate, CustomerID [FK], BookID [FK])

**3. Physical Model (PostgreSQL)**:

```
CREATE TABLE T_CUSTOMER (
  CUST_ID SERIAL PRIMARY KEY,
  CUST_NAME VARCHAR(100) NOT NULL,
  EMAIL VARCHAR(255) UNIQUE
);
CREATE TABLE T_BOOK (
  BOOK_ID SERIAL PRIMARY KEY,
  TITLE VARCHAR(500) NOT NULL,
  AUTHOR VARCHAR(200)
);
CREATE TABLE T_ORDER_ITEMS (
  ORDER_ID INT NOT NULL,
  BOOK_ID INT REFERENCES T_BOOK(BOOK_ID),
  QUANTITY INT DEFAULT 1,
  PRIMARY KEY (ORDER_ID, BOOK_ID)
);
```

# 3 OLTP vs OLAP: Two Different Worlds

Database systems are designed for fundamentally different purposes. Understanding this distinction is crucial for choosing the right modeling approach.

## 3.1 OLTP: Online Transaction Processing

**Definition:**

OLTP Systems designed for real-time operational transactions—handling many concurrent users performing fast, individual read/write operations.

**Analogy**: The **bank teller's computer**—processing deposits, withdrawals, and transfers in real-time.

**Characteristics**:

- Many users, short transactions
- Data integrity is critical (ACID compliance)
- **Normalized** data (minimize redundancy)
- Current, operational data
- Examples: ATM systems, e-commerce carts, reservation systems

## 3.2 OLAP: Online Analytical Processing

**Definition:**

OLAP Systems designed for complex analytical queries across large volumes of historical data—supporting business intelligence and decision-making.

**Analogy**: The **corporate headquarters analytics department**—analyzing years of sales data to find patterns.

**Characteristics**:

- Few users, complex queries
- Query speed is critical
- **Denormalized** data (optimize for reads)
- Historical, aggregated data
- Examples: Sales reports, customer segmentation, trend analysis

**Table 2:** *OLTP vs OLAP Systems*

| Aspect | OLTP | OLAP |
|---|---|---|
| **Purpose** | Day-to-day operations | Decision support |
| **Design** | Application-oriented | Subject-oriented |
| **Data** | Current, up-to-date | Historical, summarized |
| **Operations** | Read/Write/Update | Mostly Read (scans) |
| **Data Size** | Gigabytes | Terabytes to Petabytes |
| **Performance** | Transaction throughput | Query response time |
| **Modeling** | **ER Model (Normalized)** | **Dimensional (Denormalized)** |

### 3.3 Comparison Table

**Key Information**

**The Bridge: ETL**

ETL (Extract, Transform, Load) connects OLTP and OLAP worlds. Data flows from operational systems (OLTP) through transformation pipelines into analytical systems (OLAP), enabling business intelligence without impacting operational performance.

# 4 Dimensional Modeling for Analytics

OLAP systems use **dimensional modeling** to optimize analytical queries. This approach separates data into two categories: Facts and Dimensions.

## 4.1 Facts and Dimensions

**Definition:**

Fact Table Contains the **measurable, quantitative data**—the "what" of business events.

- Numeric measures: sales amount, quantity, clicks
- Foreign keys to dimension tables
- Very large (millions/billions of rows), narrow (few columns)

**Definition:**

Dimension Table Contains the **descriptive context**—the "who, when, where, what" of business events.

- Descriptive attributes: customer name, product category, date details
- Provides filtering and grouping criteria
- Relatively small, wide (many columns)

**Example:**

Sales Analysis **Question**: "What was total revenue by product category and region last quarter?"
**Fact Table**: Sales transactions with amount, quantity, profit
**Dimension Tables**:

- Product: category, brand, price tier
- Customer: region, segment, acquisition date
- Time: date, quarter, year, day of week

## 4.2 Star Schema

**Definition:**

Star Schema A dimensional model where one central **fact table** is surrounded by multiple **dimension tables**. The visual representation resembles a star.

**Characteristics**:

- Dimension tables are **denormalized** (contain redundant data)
- Only **one join** needed between fact and dimension
- **Fast query performance**
- Higher storage due to redundancy

### 4.3 Snowflake Schema

**Definition:**

Snowflake Schema An extension of star schema where dimension tables are **normalized** into sub-dimensions. The visual representation resembles a snowflake.

**Characteristics**:

- Dimension tables are **normalized** (separated into related tables)
- **Multiple joins** needed to traverse dimension hierarchies
- **Lower storage** due to reduced redundancy
- Slower query performance

**Table 3:** *Star Schema vs Snowflake Schema*

| **Aspect** | **Star Schema** | **Snowflake Schema** |
|---|---|---|
| Dimension tables | Denormalized | Normalized |
| Data redundancy | High | Low |
| Joins required | Few (typically 1) | Many (multiple levels) |
| Query performance | **Faster** | Slower |
| Storage efficiency | Lower | Higher |
| Complexity | Simple | Complex |

**Key Information**

**Modern Preference: Star Schema**
With cheap storage and powerful compute, most modern data warehouses prefer star schemas. The query performance benefits outweigh storage costs. Snowflake schemas are used when storage is truly constrained or when data governance requires strict normalization.

# 5 NoSQL Data Modeling

NoSQL databases require a fundamentally different mindset for data modeling.

## 5.1 The Key Difference: Query-First Design

**Warning**

**Different Questions**

- **SQL/Relational**: "Given this data structure, what questions can I answer?"
- **NoSQL**: "Given the questions business needs answered, how should I structure data?"

NoSQL is **schema-flexible**, not schema-free. The modeling is just as important—it's driven by **access patterns** rather than normalization theory.

## 5.2 Denormalization is the Norm

NoSQL databases often don't support joins (or support them poorly). Therefore:

- **Duplicate data** to avoid joins
- Structure data so **one query retrieves everything needed**
- Optimize for **read performance** over storage efficiency

## 5.3 Embedded vs Referenced Documents

**Example:**

Blog Posts and Comments **Embedded Model (Denormalized)**:

```
{
  "post_id": "p123",
  "title": "My First Post",
  "content": "Hello world!",
  "comments": [
    { "user": "alice", "text": "Great post!" },
    { "user": "bob", "text": "Welcome." }
  ]
}
```

**Pros**: One query gets everything
**Cons**: Document grows unbounded with comments
**Referenced Model (Normalized)**:

```
// Posts Collection
{ "post_id": "p123", "title": "My First Post" }

// Comments Collection
{ "comment_id": "c1", "post_id": "p123", "user": "alice" }
{ "comment_id": "c2", "post_id": "p123", "user": "bob" }
```

**Pros**: Scalable, documents stay small

**Cons**: Two queries (or expensive join) needed

**Key Information**

**Rule of Thumb**

- **Embed** when: Data is accessed together, bounded growth, 1:1 or 1:few relationships
- **Reference** when: Data accessed independently, unbounded growth, many:many relationships

# 6 Data Warehouse Architectures

Three major approaches have shaped how organizations build data warehouses.

## 6.1 Inmon (Top-Down)

**Definition:**

Inmon Approach Build a **centralized, normalized (3NF) enterprise data warehouse** first. Then create department-specific data marts from this single source of truth.

**Analogy**: "Paint the entire house first, then decorate individual rooms"

**Characteristics**:

- Third Normal Form (3NF) for the central DWH
- Data marts derived from the warehouse
- Strong data consistency and integrity
- Long initial implementation time

## 6.2 Kimball (Bottom-Up)

**Definition:**

Kimball Approach Build **department-specific data marts** (using star schema) first. The enterprise warehouse emerges as the collection of these marts connected by conformed dimensions.

**Analogy**: "Build LEGO blocks first, then assemble the structure"

**Characteristics**:

- Dimensional model (star/snowflake) from the start
- Faster initial delivery of business value
- Risk of inconsistent data across marts
- Conformed dimensions needed to maintain consistency

## 6.3 Data Vault (Hybrid)

**Definition:**

Data Vault A modeling technique designed for **flexibility and auditability**. Separates data into three components: Hubs (business keys), Links (relationships), and Satellites (attributes and history).

**Analogy**: "A vault that tracks every change to every piece of data"

**Components**:

- **Hubs**: Core business entities (e.g., CustomerID)

- **Links**: Relationships between hubs
- **Satellites**: Descriptive attributes and their change history

**Advantages**:

- Easily add new data sources
- Full audit trail of all changes
- Handles change better than Inmon/Kimball

**Table 4:** *DWH Architecture Comparison*

| **Aspect** | **Inmon** | **Kimball** | **Data Vault** |
|---|---|---|---|
| Approach | Top-down | Bottom-up | Hybrid |
| Structure | 3NF | Star schema | Hub/Link/Satellite |
| Time to value | Slow | Fast | Medium |
| Single source of truth | Yes | No | Yes |
| Handle change | Poor | Poor | **Good** |
| Complexity | Medium | Low | High |

# 7 Modern Architecture: Medallion (Bronze/Silver/Gold)

The Medallion Architecture organizes data by quality level in lakehouse environments.

## 7.1 The LEGO Analogy

**Key Information**

**Data Conformance = Building with LEGO**

Raw data is like a pile of mismatched LEGO pieces. Data engineering transforms them into:

1. **Sorted**: Organized by color/type
2. **Arranged**: Grouped logically
3. **Consistent**: Standardized sizes

The result: "Conformed data" that anyone can build with.

## 7.2 The Three Layers

**Definition:**

Bronze Layer (Raw) **Purpose**: Ingest source data with minimal transformation

**Characteristics**:

- Data as-is from sources
- Maybe add ingestion timestamp, source file name
- Preserves original for audit and reprocessing

**Definition:**

Silver Layer (Cleaned) **Purpose**: Clean, validate, and enrich data

**Characteristics**:

- Deduplicated
- Data types corrected
- Null values handled
- Business rules applied

**Definition:**

Gold Layer (Business-Ready) **Purpose**: Aggregated, business-level data for consumption

**Characteristics**:

- Star schemas and data marts
- Pre-computed aggregations
- Ready for dashboards, reports, ML

**Warning**

**Why Keep Bronze?**

Never throw away raw data! Business logic changes, new use cases emerge, and you may need to reprocess from scratch. Storage is cheap—recreating lost data is expensive or impossible.

# 8 Metadata: Data About Data

**Definition:**

Metadata Information that describes, explains, and provides context about data. It answers questions like: Who created this? Where did it come from? How should it be used?

## 8.1 Categories of Metadata

1. **Business Metadata**: Business definitions, KPIs, data ownership, glossary
2. **Technical Metadata**: Schema, data types, lineage, storage location
3. **Operational Metadata**: SLAs, refresh frequency, usage patterns, freshness

## 8.2 Key Metadata Questions

**Table 5:** *Essential Metadata Questions*

| Category | Questions |
|---|---|
| **Who** | Created? Manages? Uses? Owns? Regulates? |
| **What** | Business definition? Rules? Abbreviations? |
| **Where** | Stored? Source? Used? Backup? |
| **When** | Created? Updated? Retention period? |
| **Why** | Purpose? Business drivers? |
| **How** | Formatted? Accessed? Protected? |

## 8.3 Data Catalogs

**Definition:**

Data Catalog A centralized repository for storing and managing metadata. Enables data discovery, governance, lineage tracking, and quality monitoring.

Examples: Databricks Unity Catalog, AWS Glue Catalog, Apache Atlas

# 9 Data Formats

Choosing the right data format significantly impacts storage, performance, and compatibility.

## 9.1 Text vs Binary Formats

**Table 6:** *Data Format Comparison*

| Format | Type | Orientation | Splitable | Use Case |
|---|---|---|---|---|
| CSV | Text | Row | No | Simple interchange |
| JSON | Text | Row | No* | APIs, documents |
| Avro | Binary | Row | Yes | Streaming, schema evolution |
| Parquet | Binary | Column | Yes | Analytics, DWH |
| ORC | Binary | Column | Yes | Hive/Hadoop analytics |
| Delta | Binary | Column | Yes | Lakehouse (ACID) |

## 9.2 Row vs Column Orientation

- **Row-oriented**: Fast writes, good for OLTP
- **Column-oriented**: Fast analytical queries (aggregations), better compression

**Warning**

**Why Avoid Text Formats for Big Data?**

- Inefficient storage (numbers as strings)
- No type checking (cars in numeric fields)
- No native compression
- Not splitable (can't parallelize)
- JSON repeats metadata in every record

## 9.3 Delta Lake

**Definition:**

Delta Lake An open-source storage layer that brings **ACID transactions** to data lakes. Built on Parquet files with a transaction log.

**Key Benefits**:

- ACID transactions on data lakes
- Time travel (access previous versions)
- Efficient upserts (UPDATE + INSERT in one operation)
- Schema enforcement and evolution

- Unified batch and streaming

# 10 Data Compression

**Definition:**

Compression Encoding data using fewer bits than the original representation, reducing storage and I/O bandwidth requirements.

## 10.1 Lossy vs Lossless

- **Lossy**: Some information lost (acceptable for images, audio)
- **Lossless**: Original data fully recoverable (required for databases)

## 10.2 Common Codecs

**Table 7:** *Compression Algorithms*

| Codec | Splitable | Compression Ratio | Speed |
|---|---|---|---|
| Gzip (.gz) | No | High | Medium |
| Bzip2 (.bz2) | No | Very High | Slow |
| LZO (.lzo) | Yes | Medium | Fast |
| Snappy (.snappy) | Yes | Medium | Very Fast |
| LZ4 | Yes | Medium | Very Fast |
| Zstandard (.zst) | Yes | High | Fast |

**Key Information**

**Splitable Matters**

For distributed processing, compression format must be **splitable**—otherwise you can't parallelize across workers. Snappy and LZ4 are popular choices for big data because they're fast and splitable.

# 11 Data Profiling

**Definition:**

Data Profiling The process of examining data to understand its structure, quality, and characteristics before using it for analysis or ML.

## 11.1 Profiling Techniques

1. **Structural Discovery**: Schema consistency, data types, format correctness
2. **Content Analysis**: Min/max, mean, median, standard deviation, null counts, unique values
3. **Relationship Discovery**: Connections between datasets, entity resolution
4. **Data Correlation**: Univariate and multivariate analysis, identifying dependent variables
5. **Visualization**: Outlier detection, distribution analysis

## 11.2 Spark Data Profiling

```python
# Quick statistics
df.describe().show()

# More detailed summary
df.summary().show()

# Check nulls
from pyspark.sql.functions import col, isnan, when, count
df.select([count(when(col(c).isNull(), c)).alias(c)
           for c in df.columns]).show()
```

Listing 1: Basic Data Profiling in Spark

# 12 Lab 01: Housing Price Prediction

## 12.1 Overview

This lab walks through a complete ML workflow using housing data:

1. Load and explore data
2. Feature engineering
3. Train/test split
4. Linear regression model
5. Evaluate with RMSE

## 12.2 Key Steps

```
# Read CSV into Spark DataFrame
df = spark.read.format("csv") \
    .option("header", "true") \
    .option("inferSchema", "true") \
    .load("/path/to/housing.csv")

# Add derived columns
from pyspark.sql.functions import from_unixtime, col
df = df.withColumn("date", from_unixtime(col("date")/1000))
df = df.withColumn("zipcode", col("zipcode").cast("string"))

# Write as Delta table
df.write.format("delta").mode("overwrite").saveAsTable("housing")
```

Listing 2: Loading and Preparing Housing Data

## 12.3 Training the Model

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import numpy as np

# Split data
train_df, test_df = df.randomSplit([0.8, 0.2], seed=123)

# Train model
model = LinearRegression()
model.fit(X_train, y_train)

# Predict
predictions = model.predict(X_test)

# Evaluate
rmse = np.sqrt(mean_squared_error(y_test, predictions))
```

```
print(f"RMSE: {rmse:.2f}")
```

Listing 3: Linear Regression with scikit-learn

## 12.4 Understanding RMSE

**Definition:**

RMSE (Root Mean Square Error) A measure of the average magnitude of prediction errors. Lower is better.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

Where $y_i$ is actual value and $\hat{y}_i$ is predicted value.

**Key Information**

**Interpreting RMSE**

RMSE depends on the scale of your target variable. For housing prices ranging from $50,000 to $5,000,000, an RMSE of $50,000 might be excellent. For prices ranging $100-$500, the same RMSE would be terrible. Always consider RMSE relative to your data range.

# 13 Summary and Best Practices

**Key Summary**

**Key Takeaways**

1. **Data Modeling** progresses: Conceptual → Logical → Physical
2. **OLTP** (operational) uses normalized models; **OLAP** (analytical) uses denormalized dimensional models
3. **Star Schema**: Fast queries, more storage (denormalized dimensions)
4. **Snowflake Schema**: Less storage, slower queries (normalized dimensions)
5. **DWH Approaches**: Inmon (top-down, 3NF), Kimball (bottom-up, dimensional), Data Vault (hybrid, audit-friendly)
6. **Medallion Architecture**: Bronze (raw) → Silver (cleaned) → Gold (business-ready)
7. **NoSQL Modeling**: Query-first design, denormalize for read performance
8. **Data Formats**: Use columnar formats (Parquet, Delta) for analytics; avoid text formats for big data
9. **Compression**: Choose splitable formats (Snappy, LZ4) for distributed processing
10. **Metadata**: Essential for data discovery, governance, and trust

**Warning**

**Design Principles**

- ☐ Design a **system**, not just a schema—schemas evolve
- ☐ Start with **core business data**—don't try to model everything at once
- ☐ Document with clear **metadata**—future you will thank you
- ☐ Identify **consumption patterns**—model for how data will be queried
- ☐ Match **compression and format** to data characteristics and access patterns