CSCI E-103: Data Engineering for Analytics
Lecture 01: Introduction to Data Engineering

Harvard Extension School

Fall 2024

- **Course:** CSCI E-103: Data Engineering for Analytics
- **Lecture:** Lecture 01: Introduction
- **Instructor:** Anindita Mahapatra & Eric Gieseke
- **Objective:** Understand data engineering fundamentals, the big data ecosystem, and begin hands-on practice with Databricks

Key Summary

This lecture introduces the field of data engineering—the discipline of transforming raw data into valuable insights. We'll explore the 5 V's of Big Data (Volume, Velocity, Variety, Veracity, Value), compare OLTP vs OLAP systems, understand ACID vs BASE properties, trace the evolution from data warehouses through Hadoop to modern Spark-based platforms, and get hands-on with Databricks Free Edition for our first practical exercises.

Contents

1 Why Data Engineering Matters

1.1 The Perfect Storm: Big Data + Cloud + AI

We are living through an unprecedented convergence of three major technological forces:

- **Big Data**: Organizations now collect more data than ever before—terabytes and petabytes of information from every digital interaction
- **Cloud Computing**: On-demand compute and storage resources have democratized access to powerful infrastructure
- **Artificial Intelligence**: Machine learning models can extract patterns and make predictions, but only if fed properly prepared data

Key Information

The Analogy: Think of data as the "new oil" of the digital economy. Just as crude oil requires refineries to become useful gasoline, raw data requires data engineering pipelines to become actionable insights. The data engineer is the modern-day refinery operator.

1.2 What is Data Engineering?

Definition:

Data Engineering Data Engineering is the practice of **turning raw data into valuable insights**. It encompasses the design, construction, and maintenance of systems and infrastructure that enable the collection, storage, transformation, and delivery of data at scale.

The key phrase businesses care about is **"Speed to Insights"**—the time from when data is generated to when decision-makers can act on meaningful conclusions. Data engineers are responsible for minimizing this time.

1.3 The Data Engineering Venn Diagram

Data Engineering sits at the intersection of three disciplines:

1. **Software Engineering**: System design, distributed computing, DevOps, service deployment
2. **Data Science**: Understanding of ML algorithms, statistical modeling, business intelligence
3. **Database/Infrastructure**: Data structures, ETL pipelines, storage systems, query optimization

Example:

The Carbon to Diamond Analogy Data refinement is like transforming carbon into diamond:

1. **Raw Data**: A messy pile of carbon atoms (e.g., application log files)
2. **Sorted Data**: Carbon arranged by some criteria (e.g., logs sorted by date and user)
3. **Cleaned Data**: Impurities removed (e.g., error logs filtered, user IDs mapped)
4. **Visualized Data**: Presented in digestible form (e.g., hourly access charts)

5. **Story/Insight**: Business meaning extracted (e.g., "3 AM spike in errors correlates with specific region")

Each transformation adds value, just as pressure and time transform carbon into diamond.

2 Data Personas: Who Works with Data?

In any data-driven organization, multiple roles collaborate to extract value from data. Understanding these personas helps clarify where data engineering fits in the bigger picture.

2.1 The Five Key Data Personas

Table 1: *Data Personas and Their Responsibilities*

Role	Primary Responsibilities
Data Engineer	Builds and maintains data pipelines. Responsible for ETL (Extract, Transform, Load), data quality, and infrastructure. Creates the "plumbing" that enables all other data work.
BI Analyst	Creates dashboards and reports using SQL. Takes refined data and presents it in consumable formats for business stakeholders.
Data Scientist	Performs exploratory data analysis (EDA) and builds machine learning models. Uses statistical methods to uncover patterns and make predictions.
MLOps Engineer	Handles automation of ML pipelines. Ensures models are reliably deployed, monitored, and maintained in production.
Data Leader	Strategic role (CDO, CIO). Sets data governance policies and drives organizational data strategy.

2.2 The Hidden Complexity of ML Systems

Warning

The Iceberg Illusion: When people think of AI/ML, they often focus only on the model code—the algorithm that makes predictions. But in reality, **90% or more of a production ML system is data engineering work**.

The model is just the tip of the iceberg. Below the surface lies:

- Data collection and ingestion
- Data validation and quality checks
- Feature extraction and engineering
- Resource management and compute allocation
- Serving infrastructure
- Monitoring and alerting
- Configuration management

All of these fall within the data engineer's domain.

3 The Five V's of Big Data

Big Data isn't just "a lot of data"—it's characterized by five distinct dimensions known as the 5 V's.

3.1 Volume: The Size of Data

Definition:

Volume The physical amount of data being stored and processed, typically measured in terabytes (TB), petabytes (PB), or even exabytes (EB). When data is too large to fit on a single machine, you're dealing with Big Data's volume challenge.

Scale perspective:

- 1 Kilobyte (KB) = 1,000 bytes ≈ a short text message
- 1 Megabyte (MB) = 1,000 KB ≈ a high-resolution photo
- 1 Gigabyte (GB) = 1,000 MB ≈ an hour of HD video
- 1 Terabyte (TB) = 1,000 GB ≈ a library of 500 hours of movies
- 1 Petabyte (PB) = 1,000 TB ≈ all photos on Facebook (early 2010s)

3.2 Velocity: The Speed of Data

Definition:

Velocity How fast data is generated, collected, and processed. This ranges from batch processing (periodic bulk updates) to real-time streaming (continuous flow).

Three processing modes:

1. **Batch Processing**: Collect data over time, process in bulk at scheduled intervals (e.g., nightly reports)
2. **Micro-batch**: Small chunks processed at short intervals (e.g., every 5 minutes)
3. **Streaming/Real-time**: Data processed immediately as it arrives (e.g., fraud detection)

3.3 Variety: The Types of Data

Definition:

Variety The different forms and formats of data that must be handled. Modern systems must process structured, semi-structured, and unstructured data together.

Important:

The 80-90% Reality Approximately 80-90% of enterprise data today is **unstructured**. This includes images, audio, video, documents, and text—precisely the data that feeds modern AI models. Traditional SQL databases only handle the remaining 10-20%.

Table 2: *Data Structure Types*

Type	Characteristics	Examples
Structured	Fixed schema, rows/-columns	SQL databases, spreadsheets
Semi-structured	Schema embedded in data	JSON, XML, Parquet files
Unstructured	No predefined schema	Images, audio, video, text documents

3.4 Veracity: The Quality of Data

Definition:

Veracity The trustworthiness and accuracy of data. Can you believe what the data is telling you? This encompasses data quality, noise, bias, and completeness.

The principle **GIGO (Garbage In, Garbage Out)** applies: even the most sophisticated ML model will produce useless results if trained on poor-quality data.

3.5 Value: The Business Impact

Definition:

Value The ultimate worth of data to the business. Of all five V's, this is the most important—all the volume, velocity, variety, and veracity management is meaningless if the data doesn't create business value.

Key Information

The Value Question: Before investing in processing any dataset, always ask:

- What business question will this answer?
- Is the insight actionable?
- Does the potential value justify the processing cost?

4 Data Temperature: Hot vs Cold

Beyond the 5 V's, data engineers also think about data "temperature"—how frequently data is accessed and how urgently it needs to be processed.

4.1 The Temperature Spectrum

Definition:

Data Temperature A classification of data based on how recently it was generated and how frequently it's accessed:

- **Hot Data**: Recent, frequently accessed, requires immediate processing
- **Warm Data**: Moderate access frequency, intermediate processing needs
- **Cold Data**: Historical, rarely accessed, can be processed in batch

Example:

Temperature Examples

- **Hot**: Current stock prices, real-time fraud detection signals, live website traffic
- **Warm**: Last month's sales data, recent user activity logs
- **Cold**: Historical archives, compliance records, data from years ago

4.2 Temperature and Business Value

Warning

The Staleness Problem: As data "cools down," its value typically decreases. A fraud detection signal is most valuable in the moment it's generated. A week later, the fraudulent transaction has already cleared.

This is why businesses increasingly demand real-time processing—but real-time comes at a cost. Always ask: "Do we **really** need this in real-time? What action will we take with the insight?"

5 OLTP vs OLAP: Two Worlds of Data Processing

Understanding the distinction between transactional and analytical systems is fundamental to data engineering.

5.1 OLTP: Online Transactional Processing

Definition:

OLTP Online Transactional Processing systems handle real-time, day-to-day operations. They process individual transactions quickly—inserts, updates, deletes—and maintain the current state of business operations.

Analogy: Think of OLTP as the **cash register** at a store. Every sale, return, or exchange is recorded immediately. The system must be fast and accurate for each individual transaction.

Characteristics:

- Many small, fast transactions
- Normalized data (minimal redundancy)
- Current data (real-time state)
- Users: Front-line employees, customers
- Examples: ATM withdrawals, online shopping cart, seat reservations

5.2 OLAP: Online Analytical Processing

Definition:

OLAP Online Analytical Processing systems support complex queries across large volumes of historical data. They're optimized for reading and aggregating data, not for individual transaction updates.

Analogy: Think of OLAP as the **corporate headquarters analytics department**. They don't process individual sales—they analyze millions of past sales to find patterns like "Q3 revenue by region" or "customer churn rate over time."

Characteristics:

- Complex queries across large datasets
- Denormalized data (optimized for reading)
- Historical data (aggregated over time)
- Users: Analysts, executives, data scientists
- Examples: Quarterly sales reports, customer segmentation, trend analysis

Table 3: *OLTP vs OLAP Comparison*

Aspect	**OLTP**	**OLAP**
Purpose	Real-time transactions	Business analysis
Analogy	Cash register	Corporate analytics
Data	Current, operational	Historical, aggregated
Operations	Insert, Update, Delete	Complex SELECT queries
Users	Employees, customers	Analysts, executives
Response time	Milliseconds	Seconds to minutes

5.3 Comparison Table

5.4 ETL: Bridging OLTP and OLAP

Definition:

ETL **Extract, Transform, Load** is the process that moves data from operational (OLTP) systems to analytical (OLAP) systems:

1. **Extract**: Pull data from source systems (databases, APIs, files)
2. **Transform**: Clean, validate, aggregate, and restructure the data
3. **Load**: Write the transformed data into the analytical system

Key Information

Reverse ETL: When insights from analytical systems need to flow **back** to operational systems (e.g., ML model predictions feeding into a CRM), this is called Reverse ETL—a growing pattern in modern data architectures.

6 SQL vs NoSQL: ACID vs BASE

How data is stored depends on the consistency requirements of your use case.

6.1 ACID Properties (SQL/Relational Databases)

Definition:

ACID ACID is a set of properties guaranteeing reliable transaction processing in traditional relational databases:

- **Atomicity**: Transactions complete entirely or not at all (no partial updates)
- **Consistency**: Data always moves from one valid state to another
- **Isolation**: Concurrent transactions don't interfere with each other
- **Durability**: Once committed, data persists even after system failure

Example:

Bank Transfer (ACID Required) When transferring $1,000 from Account A to Account B:

- Account A: -$1,000
- Account B: +$1,000

Both operations must succeed or **both** must fail. If only one succeeds, money is created or destroyed—a catastrophic inconsistency. This is why financial systems require ACID compliance.

6.2 BASE Properties (NoSQL Databases)

Definition:

BASE BASE is a more relaxed consistency model for distributed NoSQL systems:

- **Basically Available**: The system guarantees availability—it will always respond
- **Soft State**: Data may be inconsistent across nodes temporarily
- **Eventual Consistency**: Given enough time, all nodes will converge to the same state

Example:

Social Media Likes (BASE Acceptable) When you "like" a post on social media:

- Your friend might see 100 likes
- You might see 101 likes
- After a few seconds, both see 101 likes

This temporary inconsistency is acceptable because:

1. It's not financially critical
2. High availability (service never down) matters more
3. Users tolerate slight delays in like counts

6.3 The CAP Theorem

Important:

CAP Theorem In any distributed data system, you can only guarantee **two out of three** properties simultaneously:

- **Consistency**: All nodes see the same data at the same time
- **Availability**: Every request receives a response (success or failure)
- **Partition Tolerance**: System continues operating despite network failures

Since network partitions are inevitable in distributed systems, the real choice is between CP (consistency + partition tolerance) and AP (availability + partition tolerance):

- **CP systems**: Traditional RDBMS, prioritize consistency
- **AP systems**: NoSQL (Cassandra, DynamoDB), prioritize availability

7 Evolution of Data Platforms

Data platforms have evolved dramatically over the past 40 years. Understanding this history helps explain why modern architectures look the way they do.

7.1 Generation 1: Data Warehouses (1980s)

- **Purpose**: Business intelligence and reporting
- **Data type**: Structured data only (SQL)
- **Technology**: Proprietary systems (Teradata, Oracle)
- **Limitations**: Expensive hardware, no unstructured data support, limited scalability

7.2 Generation 2: Hadoop and the Data Lake (2010s)

Definition:

Hadoop An open-source framework for distributed storage (HDFS) and processing (MapReduce) across clusters of commodity hardware. Originated at Yahoo in 2006.

Key innovations:

- **Commodity hardware**: Use cheap, off-the-shelf servers instead of expensive proprietary machines
- **HDFS**: Hadoop Distributed File System—store data across many machines with 3x replication
- **Data Lake**: Store **all** data (structured, semi-structured, unstructured) in raw form
- **Open source**: Free software, vibrant ecosystem

Warning

Why Hadoop/MapReduce Was Slow

MapReduce processing follows a Map → Reduce pattern. The critical problem: **every stage writes intermediate results to disk** (HDFS), then the next stage reads from disk.

Disk I/O is thousands of times slower than memory access. This made Hadoop processing painfully slow for iterative algorithms (like machine learning).

7.3 Generation 3: Apache Spark (2012+)

Definition:

Apache Spark A unified analytics engine for large-scale data processing. Originally developed at UC Berkeley's AMPLab, Spark is 10-100x faster than Hadoop MapReduce for most workloads.

Key Summary

Why Spark is Fast: In-Memory Processing

Spark's key innovation: keep data **in RAM** (memory) as much as possible, only spilling to disk when necessary.

Additionally, Spark uses:

- **DAG (Directed Acyclic Graph)**: Plans the entire computation before executing
- **Lazy Evaluation**: Transformations aren't executed until an "action" (like `count()` or `collect()`) triggers computation
- **RDD/DataFrame abstraction**: Resilient Distributed Datasets provide fault tolerance without constant disk writes

7.4 Generation 4: The Lakehouse (2020s)

Definition:

Lakehouse A modern architecture combining the best of Data Warehouses (ACID transactions, governance, reliability) with Data Lakes (flexibility, all data types, low cost). Examples: Databricks Delta Lake, Apache Iceberg.

- **From Data Warehouse**: ACID transactions, schema enforcement, data governance
- **From Data Lake**: Support for all data types, scalable cloud storage, open formats
- **Key technology**: Delta Lake adds a transaction log to Parquet files, enabling ACID on data lakes

8 Understanding Spark Architecture

Since we'll use Spark extensively through Databricks, understanding its architecture is essential.

8.1 Core Components

1. **Driver Program**: The master process that coordinates the overall execution
2. **Cluster Manager**: Allocates resources across the cluster (YARN, Kubernetes, or Spark Standalone)
3. **Worker Nodes**: Execute the actual computation tasks
4. **Executors**: JVM processes on worker nodes that run tasks and cache data

8.2 RDDs and DataFrames

Definition:

RDD (Resilient Distributed Dataset) The original Spark abstraction. RDDs are:

- **Resilient**: Can recover from node failures
- **Distributed**: Data partitioned across cluster nodes
- **Immutable**: Transformations create new RDDs, never modify existing ones

Definition:

DataFrame A higher-level abstraction built on RDDs. DataFrames organize data into named columns (like a SQL table) and enable Spark's query optimizer to generate efficient execution plans. **This is what you'll use in practice.**

8.3 Transformations vs Actions

- **Transformations**: Operations that define a new DataFrame (e.g., `select()`, `filter()`, `groupBy()`). They are **lazy**—nothing executes immediately.
- **Actions**: Operations that trigger actual computation (e.g., `count()`, `collect()`, `show()`). Only when an action is called does Spark execute the transformation pipeline.

Example:

Lazy Evaluation in Action

```
# These are transformations - nothing executes yet
df = spark.read.csv("data.csv")
df_filtered = df.filter(df["age"] > 21)
df_selected = df_filtered.select("name", "age")

# This is an action - NOW everything executes
df_selected.count()  # Triggers the entire pipeline
```

Spark waits until the `count()` action to optimize and execute the entire pipeline at once.

9 Cloud Computing Models: IaaS, PaaS, SaaS

Understanding cloud service models helps contextualize where Databricks fits.

9.1 The Pizza Analogy

Example:

Pizza as a Service Imagine you want pizza:

- **IaaS (Infrastructure as a Service)**: You get a kitchen with an oven. You buy flour, tomatoes, cheese, and make everything from scratch. Maximum control, maximum effort.
- **PaaS (Platform as a Service)**: You get pre-made dough and sauce. You just add toppings and bake. Less work, less control.
- **SaaS (Software as a Service)**: You order delivery. Pizza arrives ready to eat. Minimal effort, no control over how it's made.

Table 4: *Cloud Service Models*

Model	You Manage	Provider Manages	Examples
IaaS	OS, runtime, data, apps	Hardware, networking, virtualization	AWS EC2, Azure VMs
PaaS	Data, applications	Everything else	Heroku, Google App Engine
SaaS	Just use it	Everything	Gmail, Salesforce, **Databricks**

9.2 Databricks as SaaS

Databricks is a SaaS platform for data engineering and analytics. You don't manage:

- Cluster provisioning and scaling
- Spark installation and configuration
- Storage infrastructure
- Security patching

You focus on:

- Writing data pipelines
- Building ML models
- Creating dashboards and reports

10 The Medallion Architecture: Bronze, Silver, Gold

A common pattern for organizing data in modern lakehouse architectures.

Definition:

Medallion Architecture A multi-layer approach to organizing data by quality level:

- **Bronze Layer**: Raw data, ingested as-is from sources
- **Silver Layer**: Cleaned, validated, and enriched data
- **Gold Layer**: Aggregated, business-level data ready for consumption

10.1 Why Bronze Matters

Key Information

Never Throw Away Raw Data

The Bronze layer preserves original data because:

1. Business logic may change, requiring reprocessing
2. New use cases may need different transformations
3. Debugging issues requires access to source data
4. Compliance may require data lineage tracking

Storage is cheap. Recreating lost data is expensive or impossible.

10.2 Layer Characteristics

Table 5: *Medallion Architecture Layers*

Layer	Data State	Transformations	Users
Bronze	Raw, as-is	None or minimal	Data engineers
Silver	Cleaned, validated	Deduplication, type casting, null handling	Data engineers, analysts
Gold	Business-ready	Aggregation, joins, business logic	Business users, dashboards

11 Lab 0: Getting Started with Databricks

11.1 Databricks Free Edition Setup

Warning

Free Edition vs Trial Account

Databricks offers two types of accounts:

- **Free Edition**: Completely free forever for learning. Look for "Free Edition" logo in top-left.
- **Trial Account**: 14-day trial of enterprise features, then requires payment.

Make sure you sign up for the Free Edition! The Trial account has different features and may cause notebook loading errors.

11.2 Key Concepts in Databricks

1. **Workspace**: Your personal file system in Databricks for notebooks and folders
2. **Notebook**: An interactive document mixing code, text, and visualizations
3. **Cluster/Compute**: The processing power that runs your code
4. **Catalog**: Top-level container for organizing data (Unity Catalog)
5. **Schema**: A collection of tables within a catalog (like a database)
6. **Volume**: A path to cloud storage for files

11.3 Magic Commands

In Databricks notebooks, magic commands change the cell's language:

```
%python     # Execute Python code (default)
%sql        # Execute SQL queries
%md         # Render Markdown text
%sh         # Run shell commands
%fs         # File system operations (DBFS)
```

11.4 Your First Spark Code

```
# Path to a public dataset in Databricks
file_path = "/databricks-datasets/bikeSharing/data-001/day.csv"

# Read CSV into a Spark DataFrame
df = spark.read.format("csv") \
          .option("header", "true") \
          .option("inferSchema", "true") \
          .load(file_path)

# Display the results in a nice table
display(df)
```

Listing 1: Reading a CSV file into a DataFrame

11.5 Creating Tables with SQL

```sql
-- Set context to your catalog and schema
USE CATALOG csci_e103;
USE SCHEMA lab00;

-- Create a table from existing data
CREATE TABLE IF NOT EXISTS bike_data AS
SELECT * FROM parquet.`/databricks-datasets/samples/lending_club/parquet/`;

-- Query your new table
SELECT * FROM bike_data LIMIT 10;
```

Listing 2: Creating and querying a table

Key Information

Delta Lake is the Default

When you `CREATE TABLE` in Databricks, it automatically creates a Delta table (not regular Parquet). Delta adds:

- ACID transactions
- Time travel (access previous versions)
- Efficient updates and deletes

12 Best Practices for Data Platforms

12.1 Architectural Principles

1. **Decouple Storage and Compute**: Storage grows forever; compute should scale independently based on workload needs.
2. **Use Open Formats**: Avoid vendor lock-in by using open formats like Parquet, Delta Lake, or Iceberg. If your data is in a proprietary format, migration becomes expensive.
3. **Service-Oriented Design**: Build reusable, modular components. Each pipeline should do one thing well.
4. **Right Tool for the Right Job**: Not every problem needs the same solution. Consider trade-offs:
   - Latency requirements
   - Throughput needs
   - Access patterns
   - Cost constraints

12.2 Business Alignment

Key Information

Technology Serves Business

Every technical decision should have business justification:

- What problem does this solve?
- What's the expected ROI?
- Who is the economic buyer?
- What are the success metrics (KPIs)?

The thrill of trying new technology isn't sufficient justification. Projects without business alignment rarely make it to production.

12.3 Build vs Buy

- **Build**: Leverage in-house expertise, full control, but costs time
- **Buy**: Faster time-to-market, proven solutions, but costs money and reduces control

When competitors are catching up and you lack internal expertise, buying may be the right choice even if building would be technically possible.

13 Summary and Key Takeaways

Key Summary

What We Learned

1. **Data Engineering** transforms raw data into valuable insights—it's the "plumbing" that makes analytics and ML possible.
2. **The 5 V's of Big Data**: Volume (size), Velocity (speed), Variety (types), Veracity (quality), and Value (business impact).
3. **OLTP vs OLAP**: Transactional systems (fast, current data) vs analytical systems (complex queries, historical data). ETL bridges them.
4. **ACID vs BASE**: Strong consistency (SQL/relational) vs eventual consistency (NoSQL). Choose based on use case requirements.
5. **CAP Theorem**: In distributed systems, choose two of three: Consistency, Availability, Partition Tolerance.
6. **Platform Evolution**: Data Warehouses → Hadoop/Data Lakes → Spark → Lakehouse. Each generation solved problems of the previous.
7. **Spark's Speed**: In-memory processing + lazy evaluation + DAG optimization = 100x faster than Hadoop MapReduce.
8. **Medallion Architecture**: Bronze (raw) → Silver (cleaned) → Gold (business-ready) for progressive data refinement.
9. **Databricks**: A SaaS platform providing managed Spark, Delta Lake, and collaborative notebooks.

Warning

Action Items for This Week

☐ Create a Databricks **Free Edition** account (NOT Trial!)

☐ Import Lab 0 files into your workspace

☐ Connect to serverless compute and run the initialize notebook

☐ Complete all Lab 0 notebooks to familiarize yourself with the platform

☐ Read Chapters 1-3 of "Simplifying Data Engineering and Analytics with Delta"