Dealing with Data

Gaurav Agarwal

Data is simple. Extracting knowledge from it is complicated.

Software Engineer & Product Developer

Director of Engineering & Founder @ https://codermana.com

ex-Tarka Labs, ex-BrowserStack, ex-ThoughtWorks

What we wanted

What we got

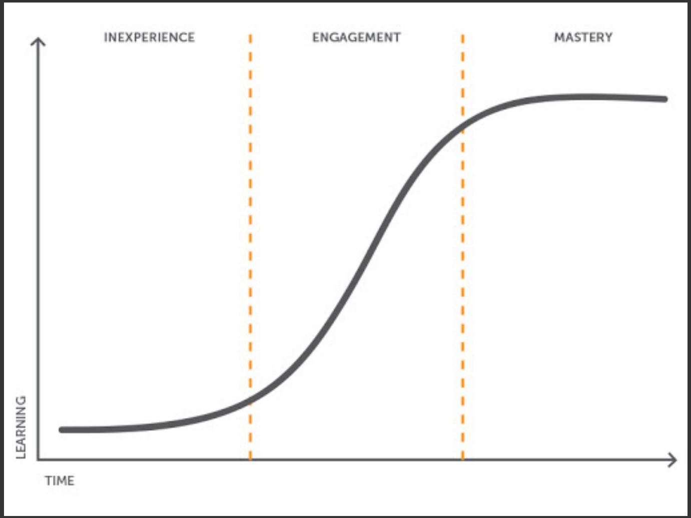
As a instructor

- I promise to
  - make this class as interactive as possible
  - use as many resources as available to keep you engaged
  - ensure everyone's questions are addressed

What I need from you

- Be vocal
    - Let me know if there any audio/video issues ASAP
    - Feel free to interrupt me and ask me questions
- Be punctual
- Give feedback
- Work on the exercises
- Be *on mute* unless you are speaking

Class progression



Here you are trying to *learn* something, while here your *brain* is doing you a favor by making sure the learning doesn't stick!

Some tips

- Slow down => stop & think
    - listen for the questions and answer
- Do the exercises
    - not add-ons; not optional
- There are no dumb questions!
- Drink water. Lots of it!

Some tips (continued)

- Take notes
    - Try: *Repetitive Spaced Out Learning*
- Talk about it out loud
- Listen to your brain
- *Experiment!*

📚 Content > 🕒 Time

Show of hands

Yay's - in Chat

Poll: What databases have you worked with?

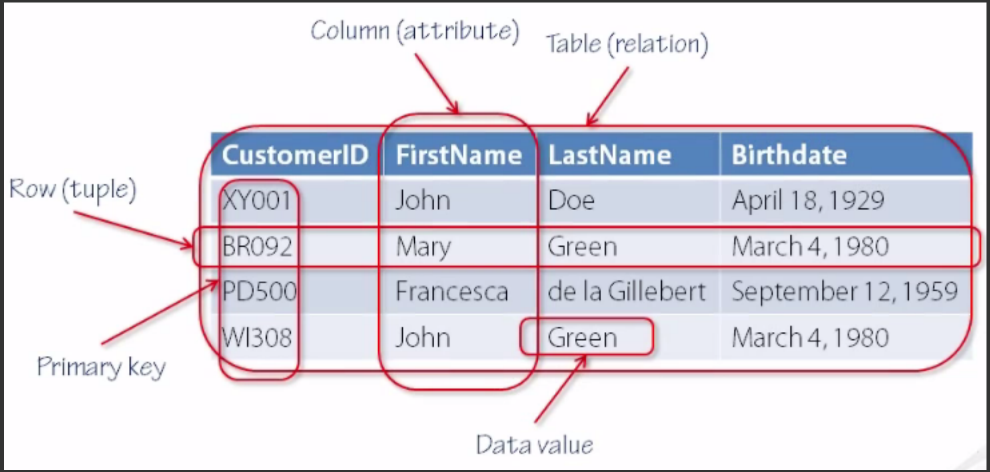
Classification of Databases

Broadly classified as: Relational, Columnar, KV, Graph, Time-series, etc.

Relational Databases

since 1970s

- multiple, related tables
- stored in rows and columns



CustomerID	FirstName	LastName	Birthdate
XY001	John	Doe	April 18, 1929
BR092	Mary	Green	March 4, 1980
PD500	Francesca	de la Gillebert	September 12, 1959
WI308	John	Green	March 4, 1980

RDBMS (Relational database management systems)

- set-theory based systems
- interacting with queries: SQL (Structured Query Language)
- Data values are typed, have schema
- tables can join

Relational Examples: `Microsoft SQL Server`, `Oracle Database`, `MySQL`, `PostgreSQL`, `IBM DB2`, `Cloud SQL`, `Spanner`, etc.

ACID!

Performance & optimization of RDBMS?

- Retrieve
- Insert
- Schema change
- Delete

What about sparse data?

Columnar Databases

aka `column data stores`

data from a given column stored together

Let's visualize...

Columnar Examples: `Google BigQuery`, `Cassandra`, `HBase`, `MariaDB`, `Azure SQL Data Warehouse`, `MonetDB`

Performance & optimization of Columnar Databases?

- Retrieve
- Insert
- Schema change
- Delete

When you’re querying a columnar database, it essentially ignores all of the data that doesn’t apply to the query.

Wide column databases

aka `column-family stores`

A Columnar data store will store each column separately on disk

A Wide-column database is a type of columnar database that supports a column family stored together on disk, not just a single column.

Ideal use cases:

- Log data
- IoT (Internet of Things) sensor data
- Time-series data, such as temperature monitoring or financial trading data
- Attribute-based data, such as user preferences or equipment features
- Real-time analytics

Wide-column store examples: `Apache Cassandra`, `Scylla`, `Apache HBase`, `Google BigTable`, `Microsoft Azure Cosmos DB`

Document Databases

store *documents*

Typically *schemaless*

- has a unique ID field
- can store nested strucutres
- can be indexed, replicated

Document Database examples: `MongoDB`, `Amazon DocumentDB`, `Apache CouchDB`, `Cloud Firestore`

Document databases are simple and scalable, making them useful for mobile apps that need fast iterations.

Key-Value Stores

simplest type of NoSQL databases

save data as a group of key-value pairs

Some KV implementations permit complex value types: hashes, list, etc.

KV Examples: `Amazon DynamoDB`, `Redis`, `Riak`, `Cloud Datastore`

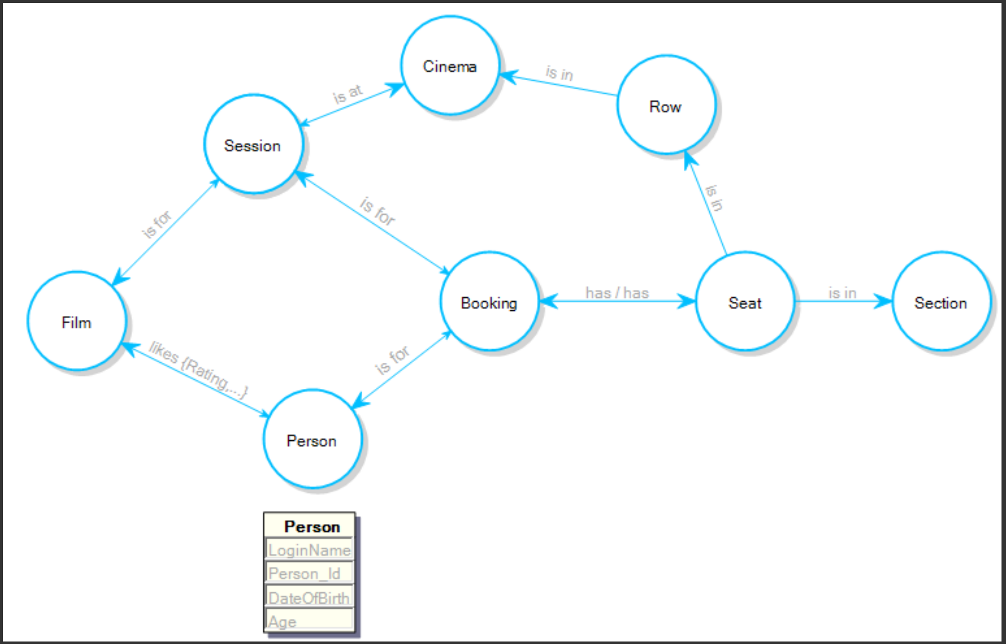
Key-value databases are highly scalable and can handle high volumes of traffic, making them ideal for processes such as session management for web applications, user sessions for massive multi-player online games, and online shopping carts.

Graph Databases

based on *graph theory*

graph databases excel at dealing with highly interconnected data

A graph database consists of nodes and relationships between nodes.



Person
LoginName
Person_Id
DateOfBirth
Age

Both nodes and relationships can have properties -- typically key-value pairs -- to store data.

Strength: traversing through nodes by following relationships

Graph database examples: `Datastax Enterprise Graph`, `Neo4J`, `Dgraph`, `ArangoDB`, `Amazon Neptune`

Time-series database

A time series database is a database optimized for time-stamped, or time series, data.

Time-series db examples: `Druid`, `eXtremeDB`, `InfluxDB`

OLTP vs OLAP

OLTP: Online Transaction Processing

An OLTP system captures and maintains transaction data in a database.

Each transaction involves individual database records made up of multiple fields or columns.

OLTP systems:

- Handles a large number of small transactions
- Simple standardized queries
- Based on `INSERT`, `UPDATE`, `DELETE` commands
- Response Time in Milliseconds
- Industry-specific, such as retail, manufacturing, or banking
- Regular backups required to ensure business continuity and meet legal and governance requirements
- Normalized databases for efficiency

OLAP: Online Analytics Processing

OLAP applies complex queries to large amounts of historical data, aggregated from OLTP databases and other sources, for data mining, analytics, and business intelligence projects.

In OLAP, the emphasis is on response time to these complex queries.

OLAP systems:

- Handles large volumes of data with complex queries
- Complex queries
- Based on SELECT commands to aggregate data for reporting
- Seconds, minutes, or hours depending on the amount of data to process
- Subject-specific, such as sales, inventory, or marketing

OLAP systems (continued):

- Purpose is to plan, solve problems, support decisions, discover hidden insights
- Data periodically refreshed with scheduled, long-running batch jobs
- Generally large due to aggregating large datasets
- Denormalized databases for analysis (Flat tables!)

The data from one or more OLTP databases is ingested into OLAP systems through a process called extract, transform, load (ETL).

Data Warehouse

A data warehouse is a type of data management system that is designed to enable and support business intelligence (BI) activities, especially analytics.

Data warehouses are solely intended to perform queries and analysis and often contain large amounts of historical data.

- centralizes and consolidates large amounts of data from multiple sources
- analytical capabilities allow organizations to derive valuable business insights to improve decision-making

Data Lake vs Data Mart

A data lake is the place where you dump all forms of data generated in various parts of your business.

Content Credits: https://cloud.google.com/learn/what-is-a-data-lake

Use Cloud Storage as Data Lake

Content Credits: https://cloud.google.com/architecture/build-a-data-lake-on-gcp

A data warehouse usually only stores data that's already modeled/structured.

Content Credits: https://cloud.google.com/architecture/bigquery-data-warehouse

A data-mart is a subsection of the data-warehouse, designed and built specifically for a particular department/business function.

3 Types of Data Mart:

- Dependent Data Marts - A dependent data mart is constructed from an existing data warehouse. It has a top-down approach that begins with storing all your business data in one centralized location, then withdraws a defined portion of the data when needed for analysis.
- Independent Data Marts - An independent data mart is a stand-alone system, which is created without the use of a data warehouse and focuses on one business function. The data is released from internal or external data sources, refined, then loaded to the data mart, where it is saved until needed or business analysis.
- Hybrid Data Marts - A hybrid data mart integrates data from a current data warehouse and additional operational source systems. It combines speed and end-user focus of a top-down approach with the assistance of the enterprise-level integration of the bottom up method.

Batch vs Stream processing

Batch Processing

Batch processing refers to processing of high volume of data in batch within a specific time span.

- It processes large volume of data all at once.
- Batch processing is used when data size is known and finite.
- It takes little longer time to processes data.
- It requires dedicated staffs to handle issues.
- Batch processor processes processes data in multiple passes.
- When data is collected overtime and similar data batched/grouped together then in that case batch processing is used.

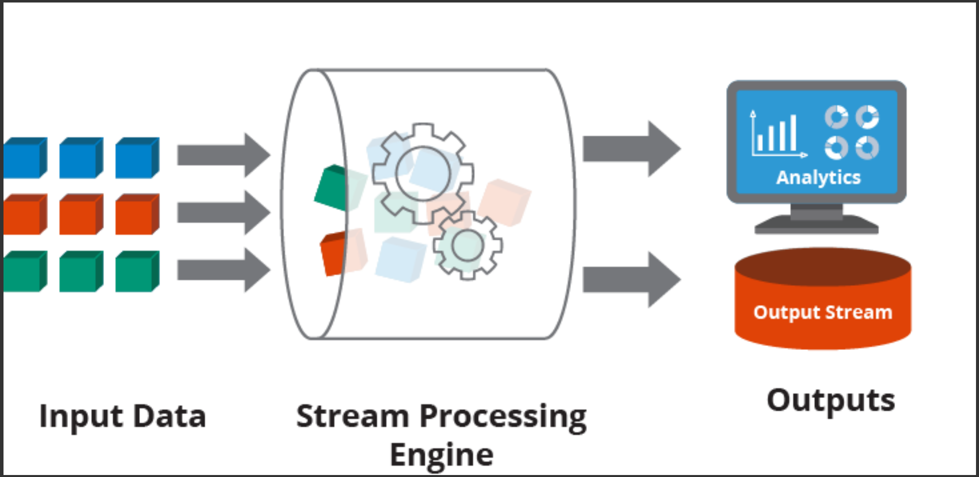
- Data is collected over time
- Once data is collected, it’s sent for processing
- Batch processing is lengthy and is meant for large quantities of information that aren’t time-sensitive

Stream Processing

Under the streaming model, data is fed into analytics tools piece-by-piece.

The processing is usually done in real time.

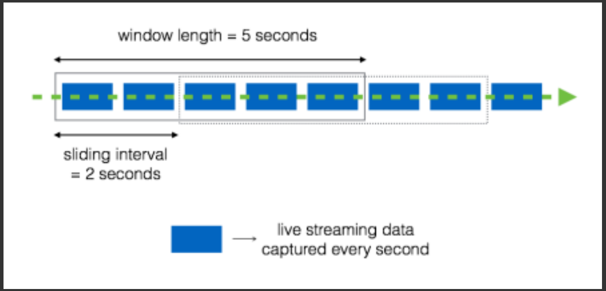
- Data streams continuously
- Data is processed piece-by-piece
- Stream processing is fast and is meant for information that’s needed immediately


Analytics
Output Stream
Input Data
Stream Processing Engine
Outputs

Stream processing has become a must-have for modern applications.

Window processing

Windowing functions divide unbounded collections into logical components, or windows.


window length = 5 seconds
sliding interval = 2 seconds
live streaming data captured every second

Stream processing examples: `Apache Spark`, `Apache Storm`

Batch vs Stream Processing

- Ingestion
  - Delivery of Data
- Processing (Eg: DataFlow (Apache Beam -> Spark, DataFlow), DataProc, CloudFunction, BigQuery, compute engine)
  - Analytics of Data
- Storage (Eg: CloudSQL, Cloud Storage, Spanner, BigTable, BigQuery, Raw-Disk/Object based Storage - like GCS)

Stream processing can be windowed too: *Fixed, Sliding*

Code https://github.com/algogrit/presentation-dealing-with-data

Slides https://dealing-with-data.slides.algogrit.com