Fundamentals of Data Engineering Chapter 7

Favorite	□
Archived	□
Fleeting	□
Area/Resource	
Project	
Type	
Review Date	
Image	
URL	
Created	@April 25, 2023 10:30 AM
Updated	@April 25, 2023 2:50 PM
Root Area	
Project Area	
Updated (short)	04/25/2023
Pulls	
Resource Pulls	
Project Archived	
URL Base	
🍳 Recipe Divider	🥗🥗🥗 RECIPE BOOK PROPERTIES 🥗🥗🥗
🍳 Recipe Tags	
📚 Book Divider	📚📚📚 BOOK TRACKER PROPERTIES 📚📚📚
📚 Author	

📅 📚 Date Started	
📅 📚 Date Finished	
⊙ 📚 Book Status	
⊙ 📚 Rating	

Data ingestion

What is data ingestion?

> 💡 Data ingestion is the process of moving data from one place to another. Data ingestion implies data movement from source systems into storage in the data engineering lifecycle, with ingestion as an intermediate step.

Data ingestion vs data integration

> 💡 Data ingestion is the data movement from point A to B, data integration combines data from disparate sources into a new dataset.

Data pipelines

> ✏️ A data pipeline is the combination of architecture, systems and processes that move data through the stages of the data engineering lifecycle.

> 💡 A data pipeline should be flexible enough to fit any needs along the data engineering lifecycle.

Key engineering considerations for the ingestion phase

Question

- What’s the use case for the data i’m ingesting?
- Can I reuse this data and avoid ingesting multiple versions of the same dataset?
- Where is the data going? What’s the destination?
- How often should the data be updated from the source?
- What is the expected data volume?
- What format is the data in? Can downstream storage and transformation accept this format?
- Is the source data in good shape for immediate downstream use? Is the data of good quality? What post-processing is required to serve it? What are data-quality risks?
- Does the data require in flight processing for downstream ingestion if the data is from a streaming source?

Considerations

- Bounded versus unbounded
- Frequency
- Synchronous versus asynchronous
- Serialization versus deserialization
- Throughput and scalability
- Reliability and durability
- Payload
- Push versus pull patterns

Bounded versus unbounded

💡 Unbounded data is data as it exists in reality, as events happen, either sporadically or continuously, ongoing and flowing. Bounded data is a convenient way of bucketing data across some sort of boundary, such as time.

> All data is unbounded until it’s bounded

⚙️ A grocery list was written as a stream of consciousness (unbounded data) onto a piece of scrap paper, where the thoughts now exist as a list of things (bounded data).

💡 Streaming ingestion systems are simply a tool for preserving the unbounded nature of data so that subsequent steps int he lifecycle can also process it continuously.

Frequency

- Batch
- Micro batch
- Real-time

💡 No pipeline is genuinely real time. Any database, queue or pipeline has inherent latency in delivery data to a target system. ML models are typically trained on a batch basis, although continuous online training is becoming more prevalent. Rarely can data engineers build purely near real-time pipelines with no batch components, they choose where batch boundaries will occur. The batch frequency will become a bottleneck for all down-stream processing.

💡 In IOT applications, the typical pattern is for each sensor to write events or measurements to streaming systems as they happen. While this could be written directly to a database a streaming ingestion platform such as Kinesis or Kafka is a better fit for the application. Software applications can adopt similar patterns by writing events to a message queue as they happen rather than waiting for an extraction process to pull events from a back end database.

Synchronous vs asynchronous ingestion

💡 With synchronous ingestion, the source, ingestion and destination have complex dependencies and are tightly coupled. Process C depends on process B which depends on A. If A fails everything else fails.

💡 With asynchronous ingestion, dependencies can now operate at the level of individual events, much as they would in a software back end build from micro services.

💡 The primary idea is that each stage of the processing pipeline processes data items as they become available in parallel.

✏️ A buffer is a stage in a DE pipeline that acts a shock absorber. It is a temporary staging area that allows stages to be decoupled, and allows each stage to operate at it's own pace based on available compute.

Serialization and Deserialization

💡 Moving data from source to destination involves serialization and deserialization. Serialization means encoding data from a source and preparing data structures for transmission and intermediate storage stages. The destination must have a means to properly deserialize the data.

Throughput and scalability

💡 Data ingestion should never be a bottleneck. Data throughput and system scalability become critical as your data volumes grow and requirements change. Design your system to scale and shrink to flexibly match the desired data throughput. Use managed services that handle the throughput scaling for you.

⚠️ Suppose a source database goes down. When it comes back online and attempts to back-fill the lapsed data loads, will your ingestion be able to keep up with the sudden influx?

Reliability and durability

💡 Reliability entails high up time and proper fail-over for ingestion systems. Durability entails making sure that data isn't lost or corrupted. Reliability of ingestion systems leads directly to the durability of generated data. Continuously evaluate the trade-offs and costs of reliability and durability.

Payload

💡 A payload is the dataset you're ingesting and has characteristics such as kind, shape, size, schema and data types, and metadata.

Kind

> 💡 Kind consists of type and format. Type can be tabular, image, video, text. Format determines the way it is expressed in bytes, names and file extensions. CSV or Parquet

Shape

> 💡 Every payload has a shape that describes its dimensions. RGB for images.

- Tabular
  - Number of rows and columns
- Semi-structured JSON
  - The key-value pairs and nesting depth occur with sub-elements.
- Unstructured text
  - Number of words, characters or bytes in the text body
- Images
  - RGB, height, width
- Uncompressed audio
  - number of channels

Size

> 💡 The size describes the number of bytes of a payload. To reduce the size it may be compressed into various formats such as ZIP and TAR. Can also be split into chunks.

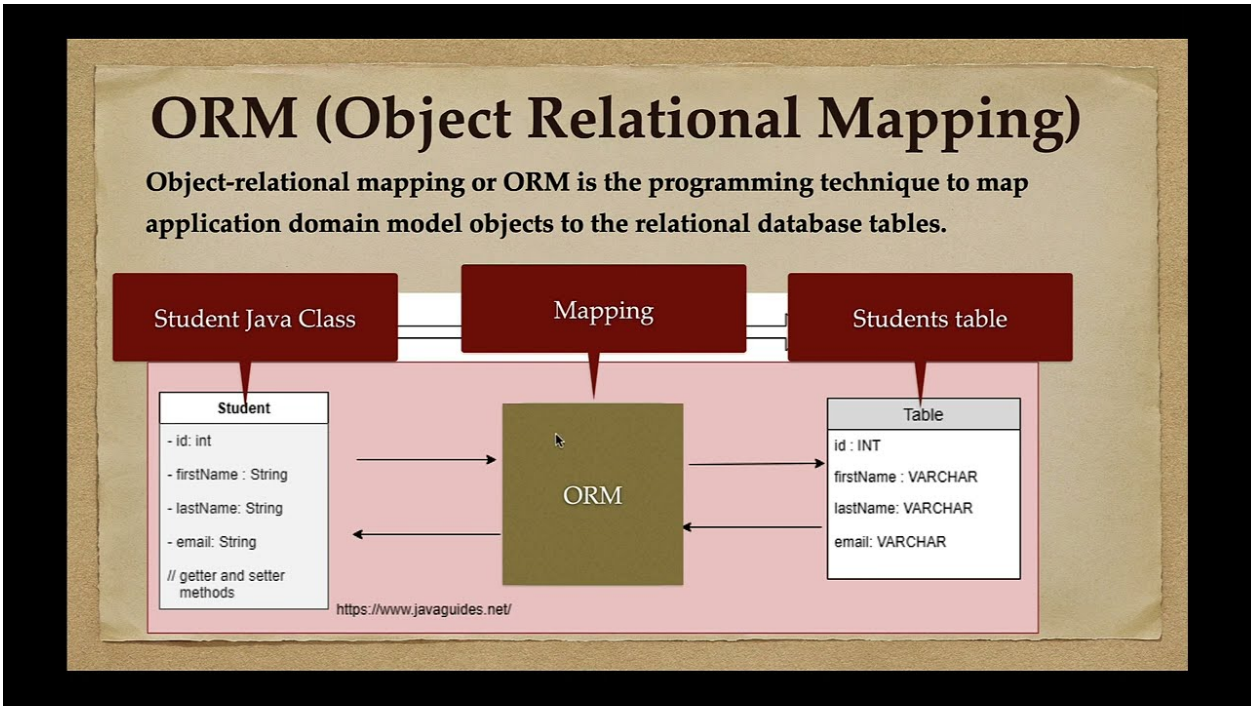
Schema and data types

💡 A schema describes the fields and types of data within those fields. This depends on the format and type of data. Understanding the schema is a great challenge. Applications organize data in various ways, and engineers need to be familiar with the organization of the data and relevant update patterns to make sense of it.

✏️ JDBC stands for Java DataBase Connectivity. It is an API for java that allows for accessing relational databases in Java.

ORM

✏️ Object Relational Mapping is a programming technique that allows developers to map their code objects directly to tables in a database.



ORM (Object Relational Mapping)

Object-relational mapping or ORM is the programming technique to map application domain model objects to the relational database tables.

> ⚠️ Engineers should study source systems as soon as they to plan to ingest data from a new source.

Changes in schemas

> 💡 Changes in schemas happen frequently. It is more common to make ingestion tools to automate the detection of schema changes and even auto-update target tables. Schema changes can still break pipelines downstream of staging and ingestion. This makes communication crucial when a schema changes to avoid a unseen impact on downstream processes.

- Adding a new column
- Changing a column type
- Creating a new table
- Renaming a column

Schema registries

> 💡 A schema registry is a metadata repository used to maintain schema and data type integrity in the face of constantly changing schemas. Schema registries can also track schema versions and history. It describes the data model for messages, allowing consistent serialization and deserialization between producers and consumers.

Metadata

> 💡 Metadata can be just as critical as the data itself. One of the significant limitations of the early approach to the data lake or data swamp which become a data super fund site was a complete lack of attention to metadata. Without a detailed description of the data it may be of little value.

Pull Versus Pull Versus Poll patterns

> 💡 A push strategy involves a source system sending data to a target, while a pull strategy entails a target reading data directly from a source.

Polling

> 💡 Polling for data involves periodically checking a data source for any changes. When changes are detected, the destination pulls the data as it would in a regular pull situation.

Batch ingestion considerations

> 💡 Data is ingested by taking a subset of data in a source system, based either on a time interval or the size of the accumulated data. Ingestion patterns are below.

- Snapshot or differential extraction
- File based export and ingestion
- ETL versus ELT
- Inserts, updates and batch size
- Data Migration

Snapshot or differential extraction

> 💡 DEs must choose to capture the full snapshots of a source system or differential (incremental) updates. Full snapshots grab the entire current state of the source system on each update read. Incremental update patter, only changes since last read are pulled.

File based export and ingestion

> 💡 Data is quite often moved between databases and systems using files. Data is serialized into files in an exchangeable format, and these files are provided to an ingestion system. This is considered a push-based pattern. This is because data export and preparation is done on the source system side.

ETL versus ELT

Extract

> 💡 Getting data from a source system. While extract seems to imply pulling data, it can also be pushed based. It may also require reading metadata and schema changes.

Load

> 💡 Once extracted data can either be transformed before loading it into a storage destination or simply loaded intro storage for future transformation.

Insert, updates and batch size

> 💡 Batch-oriented systems perform poorly when users attempt to perform many small-batch operations rather than a smaller number of large operations. BigQuery performs poorly on a high rate of vanilla SQL single-row inserts but extremely well if data is fed in through it’s steam buffer. Know the limits and characteristics of your tools.

Data migration

> 💡 Most data systems perform best when data is moved in bulk rather than as individual rows or events. File or object storage is often an excellent intermediate stage for transferring data.

Message and stream ingestion considerations

Schema evolution

> 💡 Schema evolution is common when handling event data; fields may be added or removed, or value types might change. Schema evolution can have unintended impacts on your data pipelines.

Schema registry

> 💡 If your event-processing framework has a schema registry, use it to version your schema changes.

- □ Look into Dead letter Queues
    - □ Holds messages that failed to process

Communication

💡 Have an open communication stream with downstream stakeholders to allow them to address changes.

Late arriving data

💡 Some data may occur at certain time but arrive later. To handle late arriving data you need to set up a cutoff time when late-arriving data is no longer processed.

Ordering and multiply delivery

💡 Streaming platforms are generally built out of distributed systems, which can cause some complications. Messages may be delivered out of order and more than once.

Replay

💡 Replay allows readers to request a range of messages from the history, allowing you to rewind your event history to a particular point in time. (Seeking). It is useful when you need to re-ingest and reprocess data for a specific time range.

Time to Live

> 💡 Time to Live TTL is a key parameter that determines the maximum message retention time. It is a configuration you'll set for how long you want a message to live before they are acknowledged and ingested. Long TTL will cause a large backlog, short TTL will cause messages to be missed.

Message size

> 💡 Make sure the platform can handle the maximum expected message size.

Error handling and dead-letter queues

> 💡 Messages that couldn't be handled need to be sent to dead-letter queues for future analysis, and diagnosis, they could use the replay to fix the error messages. If messages aren't sent to a dead letter queue these messages risk blocking other messages from being ingested.

Consumer Pull and Push

Pull

> 💡 A consumer subscribing to a topic can get events by pulling data. Subscribers read messages from a topic and confirm when they have been processed.

Push

> 💡 Pub/Sub and RabbitMQ allow services to write messages to a listener.

Location

> 💡 It is often desirable to integrate streaming across several locations for enhanced redundancy and to consumer data close to where it is generated. Generally, the closer to data origination the better the bandwidth and latency. Balance data egress costs.

Ways to ingest data

Direct database connection

> 💡 Data can be pulled from database for ingestion by querying and reading over a network connection. Most commonly this connection is made using ODBC and JDBC.