Data Engineering

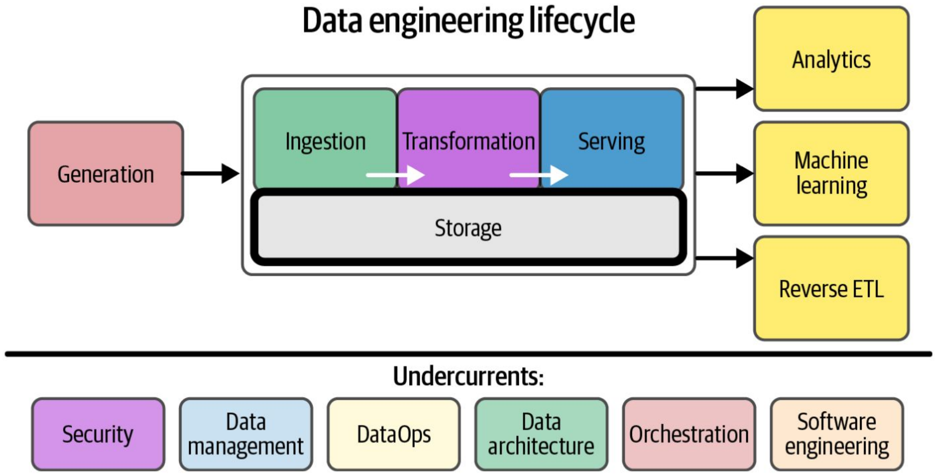
Saturday, 17 Dec 2022

Course content

1. Introduction to Data Engineering
2. The difference between other data functions
3. Fundamental SQL & NoSQL query
4. Fundamental distributed system
5. Fundamental Unix based system & containerization
6. Fundamental database (OLTP & OLAP)
7. Fundamental Python & Scala for data processing
8. Use Pandas & Spark to ingest, transform, clean and prepare data
9. Perform data quality testing with Python
10. Create end to end data pipeline using Apache Airflow scheduler Orchestration, and workflow engine

Storage

Storage is a place to store the data. Let's recall our previous material about the data engineering lifecycle. Storage underlies major stages - ingestion, transformation, and serving. To paraphrase an old saying, it's storage all way down. Knowing the use case of the data and the way we will retrieve it in the future is the first step to choosing the proper storage solutions for our data architecture.

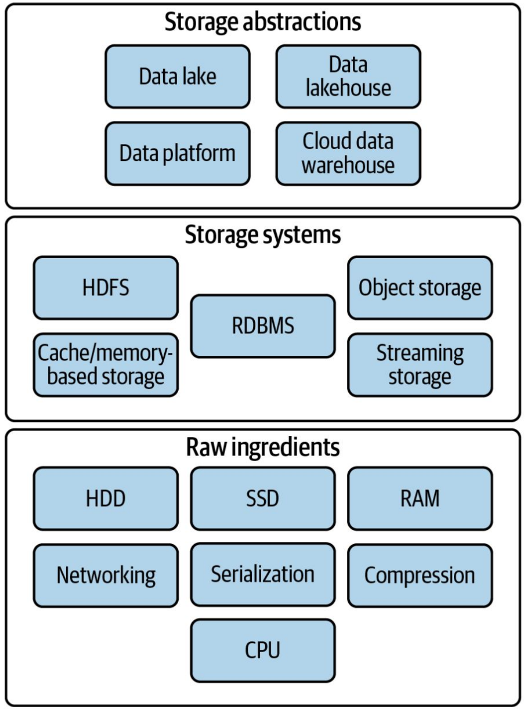


Raw ingredients, storage systems, and storage abstractions

Raw ingredients – compose storage systems, including hard drives, solid-state drives, and system memory. It's essential to understand the basic characteristics of physical storage technologies to assess the trade-offs inherent in any storage architecture.

Storage systems – In practice, we don't directly access system memory or hard disks. These physical storage components exist inside servers and clusters that can ingest and retrieve data using various access paradigms.

Storage abstractions – Storage systems are assembled into a cloud data warehouse, a data lake, etc. When building data pipelines, engineers choose the appropriate abstractions for storing their data as it moves through the ingestion, transformation, and serving stages.



SQL vs NoSQL

SQL (Structured Query Language) is a programming language used to communicate with databases. It is used to create, modify, and query databases. SQL is used to retrieve and manipulate data stored in a relational database management system (RDBMS).

NoSQL (not only SQL) refers to a type of database that does not (only) use the SQL language to manipulate and query data. NoSQL databases are designed to handle large amounts of data that may not be structured in a traditional, tabular relational database. NoSQL databases are often used in big data and real-time web applications. There are several types of NoSQL databases, including document databases, key-value stores, graph databases, and column-oriented databases.

SQL & RDBMS

Relational database management systems (RDBMS) are a type of database management system that is based on the relational model. They store data in the form of tables, with rows representing individual records and columns representing the attributes of those records. RDBMSs use the Structured Query Language (SQL) to create, modify, and query the data stored in the database.

Here are some examples of RDBMSs:

- MySQL: MySQL is an open-source RDBMS that is widely used for web applications and data storage. It is known for its speed, reliability, and simplicity.
- Oracle: Oracle is a commercial RDBMS developed by Oracle Corporation. It is widely used in enterprise environments and is known for its scalability, security, and performance.
- Microsoft SQL Server: SQL Server is a commercial RDBMS developed by Microsoft. It is designed to be used with the Windows operating system and is often used in business and enterprise environments.
- PostgreSQL: PostgreSQL is an open-source RDBMS that is known for its robustness, flexibility, and support for advanced features such as stored procedures and triggers.

NoSQL

Here are some examples of different types of NoSQL databases:

Document databases: Document databases store data in the form of documents, which are similar to JSON objects. Document databases are designed to be flexible and scalable, and they are often used to store large amounts of unstructured data. Examples of document databases include MongoDB and Couchbase.

Key-value stores: Key-value stores store data as a mapping of keys to values. They are designed to be simple and fast, and they are often used to store large amounts of data that needs to be quickly retrieved by a unique key. Examples of key-value stores include Redis and DynamoDB.

Graph databases: Graph databases store data in the form of nodes and edges, which represent entities and relationships between entities. They are designed to efficiently store and query data about relationships and connections. Examples of graph databases include Neo4j and Amazon Neptune.

Column-oriented databases: Column-oriented databases store data in columns rather than rows. They are designed to be efficient at storing and querying large amounts of data, and they are often used for data warehousing and analytics. Examples of column-oriented databases include Cassandra and HBase.

Best Use Case for SQL / RDBMS

SQL (Structured Query Language) databases, also known as relational database management systems (RDBMS), are well-suited for storing structured data that can be organized into tables with rows and columns. They are particularly good at handling complex queries, transactions, and foreign key relationships, which are connections between different tables in the database.

SQL databases are often used in applications where data integrity and consistency are important, such as financial systems, medical records, and e-commerce websites. They are also well-suited for data that needs to be accessed and updated frequently, as they offer fast read and write speeds.

Best Use Case for NoSQL

On the other hand, NoSQL (not only SQL) databases are well-suited for storing large amounts of data that may not be structured in a traditional, tabular manner. They are designed to handle data that is distributed across multiple servers and can scale horizontally, meaning they can easily handle an increase in the volume of data or the number of users accessing the database.

NoSQL databases are often used in big data and real-time web applications, such as social media platforms, online gaming, and IoT (Internet of Things) applications. They are also well-suited for handling large amounts of unstructured data, such as text, images, and video.

In general, the choice between a SQL and a NoSQL database will depend on the specific needs of the application and the type of data being stored. Both types of databases have their own strengths and can be used effectively in different situations.

SQL/RDBMS Use Case Scenario (PostgreSQL)

Here is a simple example of a use case for a RDBMS in PostgreSQL:

Imagine you own a small retail store and you want to keep track of your inventory using a database. You decide to use PostgreSQL as your RDBMS.

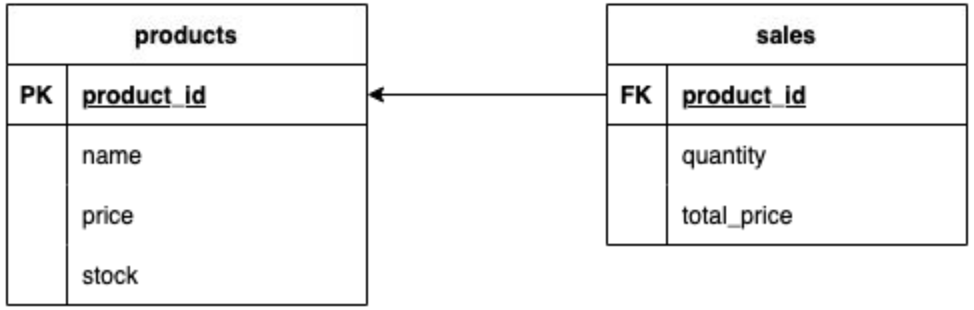
First, you create a table called "products" to store information about each product in your inventory. The table has columns for the product's name, price, quantity in stock, and a unique identifier called a "product_id".

Next, you create a table called "sales" to store information about each sale made at your store. The table has columns for the sale's date, the product_id of the item sold, the quantity sold, and the total price.

Using SQL commands, you can insert rows into these tables to add new products and sales to the database. You can also use SQL commands to retrieve and update information in the tables, such as updating the quantity in stock when a product is sold or retrieving a list of all sales made on a particular date.

With these tables and SQL commands, you can easily track and manage your inventory and sales using a RDBMS in PostgreSQL.

Entity Relational Diagram



NoSQL Use Case Scenario (MongoDB)

Here is a simple example of a use case for a NoSQL database in MongoDB:

Imagine you are building a social media platform and you want to store user profiles in a database. You decide to use MongoDB as your NoSQL database.

First, you create a collection called "users" to store information about each user on your platform. The collection has documents with fields for the user's name, email address, password, and a unique identifier called a "user_id".

Next, you create a collection called "posts" to store information about each post made by users on your platform. The collection has documents with fields for the post's content, the user_id of the user who made the post, the date the post was made, and the number of likes the post has received.

Using MongoDB commands, you can insert documents into these collections to add new users and posts to the database. You can also use MongoDB commands to retrieve and update information in the collections, such as updating the number of likes a post has received or retrieving a list of all posts made by a particular user.

With these collections and MongoDB commands, you can easily store and manage user profiles and posts using a NoSQL database in MongoDB.

JSON Format

```
// users
{
  "_id": "",
  "user_id": "",
  "name": "",
  "email_address": "",
  "password": ""
}
```

```
// posts
{
  "_id": "",
  "content": "",
  "published_at": "",
  "likes": 0,
  "user_id": ""
}
```