Data modeling
Data modeling in software engineering is the process of creating a data model for an information system by applying certain formal techniques.
The Correct Order of the Data Modeling Process: conceptual -> logical -> physical
Data modeling is an iterative process but not a fixed one.

Relational databases
Relational model: relational model organizes data into one or more tables(relations) of columns(attributes) and rows(tuples), with a unique key identifying each row.
Relational database: a digital database based on the relational model of data.
RDBMS: a software system used to maintain relational databases is a relational database management system.
Common types of relational databases: PostgreSQL, MySQL, Teradata, Oracle, Sqlite

Advantages of Using a Relational Database

- Flexibility for writing in SQL queries: With SQL being the most common database query language.
- Modeling the data not modeling queries
- Ability to do JOINS
- Ability to do aggregations and analytics
- Secondary Indexes available : You have the advantage of being able to add another index to help with quick searching.
- Smaller data volumes: If you have a smaller data volume (and not big data) you can use a relational database for its simplicity.
- ACID Transactions: Allows you to meet a set of properties of database transactions intended to guarantee validity even in the event of errors, power failures, and thus maintain data integrity.
- Easier to change to business requirements

ACID Transactions: Properties of database transactions intended to guarantee validity even in the event of errors or power failures.

- Atomicity**:** The whole transaction is processed or nothing is processed. A commonly cited example of an atomic transaction is money transactions between two bank accounts. The transaction of transferring money from one account to the other is made up of two operations. First, you have to withdraw money in one account, and second you have to save the withdrawn money to the second account. An atomic transaction, i.e., when either all operations occur or nothing occurs, keeps the database in a consistent state. This ensures that if either of those two operations (withdrawing money from the 1st account or saving the money to the 2nd account) fail, the money is neither lost nor created. Source Wikipedia for a detailed description of this example.
- Consistency: Only transactions that abide by constraints and rules are written into the database, otherwise the database keeps the previous state. The data should be correct across all rows and tables. Check out additional information about consistency on Wikipedia.

- Isolation: Transactions are processed independently and securely, order does not matter. A low level of isolation enables many users to access the data simultaneously, however this also increases the possibilities of concurrency effects (e.g., dirty reads or lost updates). On the other hand, a high level of isolation reduces these chances of concurrency effects, but also uses more system resources and transactions blocking each other. Source: Wikipedia
- Durability: Completed transactions are saved to database even in cases of system failure. A commonly cited example includes tracking flight seat bookings. So once the flight booking records a confirmed seat booking, the seat remains booked even if a system failure occurs. Source: Wikipedia.

When Not to Use a Relational Database

- Have large amounts of data: Relational Databases are not distributed databases and because of this they can only scale vertically by adding more storage in the machine itself. You are limited by how much you can scale and how much data you can store on one machine. You cannot add more machines like you can in NoSQL databases.
- Need to be able to store different data type formats**:** Relational databases are not designed to handle unstructured data.
- Need high throughput -- fast reads**:** While ACID transactions bring benefits, they also slow down the process of reading and writing data. If you need very fast reads and writes, using a relational database may not suit your needs.
- Need a flexible schema**:** Flexible schema can allow for columns to be added that do not have to be used by every row, saving disk space.
- Need high availability: The fact that relational databases are not distributed (and even when they are, they have a coordinator/worker architecture), they have a single point of failure. When that database goes down, a fail-over to a backup system occurs and takes time.
- Need horizontal scalability**:** Horizontal scalability is the ability to add more machines or nodes to a system to increase performance and space for data.

PostgreSQL: an open-source object-relational database system. It uses and builds on SQL language.

Demo:

```
import psycopg2 as py
try:
        conn = py.connect("host= …. dbname=… user=… password=…")
except py.Error as e:
        print("e")
try:
        cur = conn.cursor()
except py.Error as e:
        print("e")
conn.set_session(autocommit = True)
cur.execute("CREATE database XXX")
```

```
cur.execute("CREATE TABLE IF NOT EXISTS XXX ( xx varchar, xx int, …)")
cur.execute("SELECT * FROM  XXX")
cur.execute("INSERT INTO XXX (xx, xx, ..) VALUES (%s, %s, …), ("…", …)"  )
print(cur.fetchall())
row = cur.fetchone
cur.close()
conn.close()
```

NoSQL database: Not only SQL, has a simpler design, simpler horizontal scaling, and finer control of availability. Data structures used are different than those in relational database are make some operations faster. NoSQL and NonRelational are interchangeable terms. There are various types of NoSQL databases, such as Apache Cassandra (Partition row store), MongoDB (document store), DynamoDB (key-value store), Apache HBase (Wide Column Store), Neo4J (Graph database)

Basic of Apache Cassandra: keyspace – collection of tables; table – a group of partitions; rows – a single item; partition – fundamental unit of access, collection of rows, how data is distributed; primary key – primary is made up of a partition key and clustering columns; columns – clustering and data, labeled elements

Demo:

```
import cassandra
from cassandra.cluster import Cluster
try:
        cluster= Cluster(['…'])
        session = cluster.connect()
except Exception as e:
        print(e)

try:
        session.execute(""""
        CREATE KEYSPACE IF NOT EXISTS xxx
        WITH REPLICATION =
        {'class':'SimpleStrategy', 'replication_factor':1}
        """")
except Exception as e:
        print(e)

try:
        session.set_keyspace('…')
except Exception as e:
        print(e)

query = "CREATE TABLE IF NOT EXIST XXX"
```

```
query = query + “(year int, name text, …, PRIMARY KEY(year, name))”
try:
        session.execute(query)
except Exception as e:
        print(e)

query = “INSERT INTO xxx (….)”
query = query + “ VALUES (%s, …)”
try:
        session.execute(query)
except Exception as e:
        print(e)
```