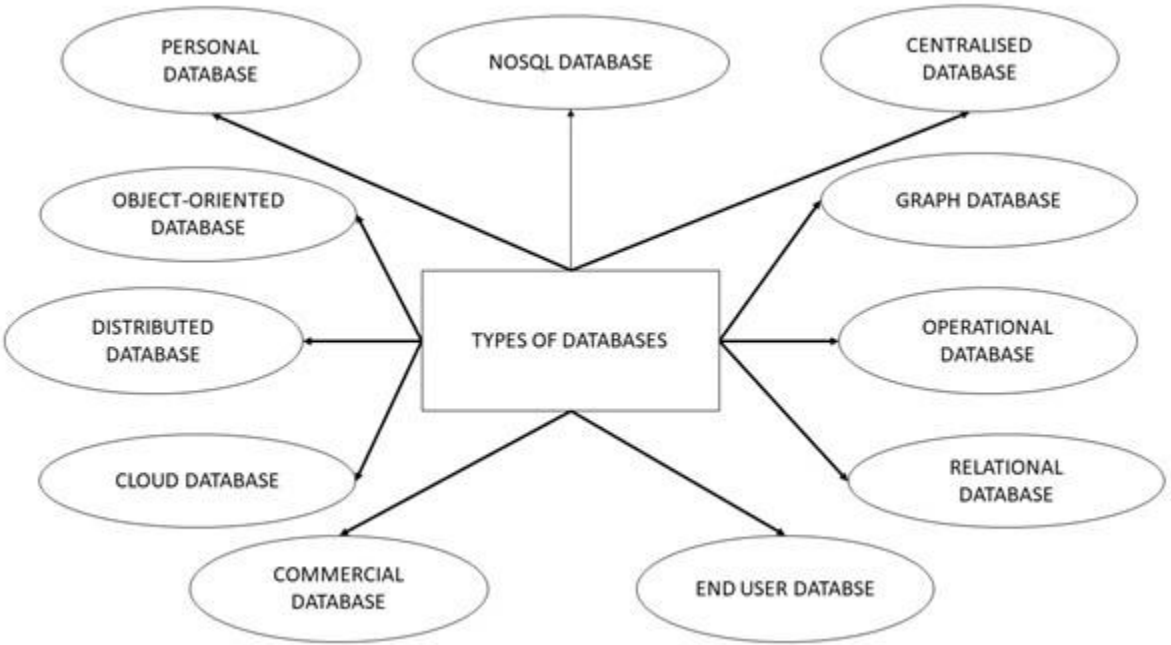
Introduction to Spring

Data persistence

Definition

- Non-persistent
  - In memory
- Persistent
  - Local files
  - Databases

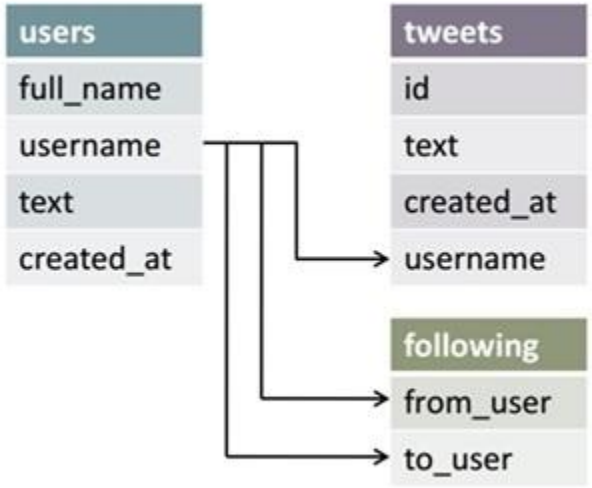
Types of databases



Relational databases

- These databases are categorized by a set of tables where data gets fit into a pre-defined category. The table consists of rows and columns where the column has an entry for data for a specific category and rows contains instance for that data defined according to the category. The Structured Query Language (SQL) is the standard user and application program interface for a relational database.
- There are various simple operations that can be applied over the table which makes these databases easier to extend, join two databases with a common relation and modify all existing applications.

Relational databases



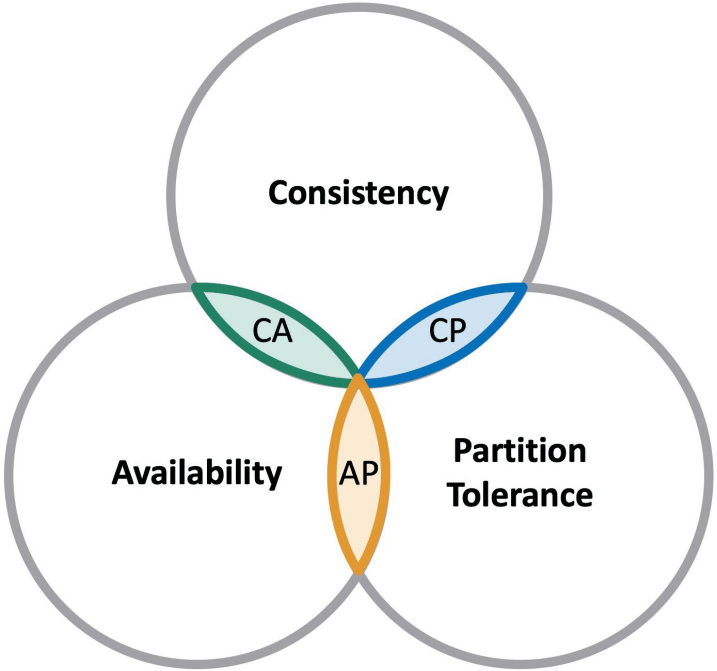
NoSql databases

- These are used for large sets of distributed data.
- There are some big data performance issues which are effectively handled by relational databases, such kind of issues are easily managed by NoSQL databases.
- There are very efficient in analyzing large size unstructured data that may be stored at multiple virtual servers of the cloud.

CAP Theorem

- The CAP theorem (also called Brewer’s theorem) states that a distributed database system can only guarantee two out of these three characteristics:
  - Consistency
  - Availability
  - Partition Tolerance.

CAP Theorem



NoSQL databases

- NoSQL database is used for large sets of distributed data.
- There are a few big data performance problems that are effectively handled by relational databases.
- This type of computers database is very efficient in analyzing large-size unstructured data.

JPA

Java Persistence API (JPA)

- The high number of ORM instruments lead to the need of a standard ●
- JPA
  - Collection of classes and methods to persistently store the vast amounts of data into a database
  - It forms a bridge between object models (Java program) and relational models (database program)
  - Based on entities

Entities

- Are POJOs (Plain Old Java Objects)
- Conventions:
  - Default constructor
  - Getters and setters for non - Boolean properties
  - Setter and is methods for Boolean properties
- Can be defined through XML or Annotations

MySql

MySql

- Enables you to implement a database with tables, columns and indexes.
- Guarantees the Referential Integrity between rows of various tables.
- Updates the indexes automatically.
- Interprets an SQL query and combines information from various tables.

Cheat sheet

- Database − A database is a collection of tables, with related data.
- Table − A table is a matrix with data. A table in a database looks like a simple spreadsheet.
- Column − One column (data element) contains data of one and the same kind, for example the column postcode.
- Row − A row (= tuple, entry or record) is a group of related data, for example the data of one subscription.

Cheat sheet

- Redundancy − Storing data twice, redundantly to make the system faster.
- Primary Key − A primary key is unique. A key value can not occur twice in one table. With a key, you can only find one row.
- Foreign Key − A foreign key is the linking pin between two tables.

Cheat sheet

- Compound Key − A compound key (composite key) is a key that consists of multiple columns, because one column is not sufficiently unique.

- Index − An index in a database resembles an index at the back of a book.

- Referential Integrity − Referential Integrity makes sure that a foreign key value always points to an existing row.

Configure mysql in spring

- Add dependency in maven
- Configure the application.properties
- Add repository and data model

Maven

```
<dependencies>
  <!-- JPA Data (We are going to use Repositories, Entities, Hiberna
  <dependency>
    <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
  <!-- Use MySQL Connector-J -->
   <dependency>
      <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
    </dependency>
<dependencies>
```

Copy

1

Applicatin.properties

```
spring.datasource.url=jdbc:mysql://localhost:3306/javadevjournal
spring.datasource.username=javadevjournal
spring.datasource.password=ThePassword
spring.jpa.hibernate.ddl-auto=update

## Hibernate Properties
# The SQL dialect makes Hibernate generate better SQL for the chosen
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQ
```

Object Relational Mapping (ORM)

- Java – objects
- Database – relational model
- ORM – the process of translating the information between the objectual model and the relational one
- Purpose – data management through Java objects

ORM

- ORM – automatic persistence of data
- Translation between the objectual model and the relational model
- Software
  - Hibernate
  - Oracle TopLink
  - MyBatis
  - EclipseLink
  - OpenJPA
  - Apache Cayenne

PROs / CONs

- Advantages ○
  - Easy to use and understand
  - Reduces the implementation time
  - Reduces the amount of code and the error rate
  - The application is independent of the database management system
- Disadvantages
  - Lower performance that manually writing the SQL queries regarding big data processing
  - Ramp - up time

Entities annotation (1)

- @Entity - declare the class as entity or a table
- @Table - declare table name
- @Id - specifies the property uses for identity (primary key of a table)
- @GeneratedValue - specifies how the identity attribute can be initialized such as automatic, manual, or value taken from sequence table
- @Transient - specifies the property won’t persist
- @Column - specifies column or attribute for persistence property
- @UniqueConstraint - used to specify a unique constraint on the field

Entities annotation (2)

- @JoinColumn - specify an entity association or entity collection. This is used in many - to - one and one - to - many associations
- @ManyToMany - define a many - to - many relationship between the join tables
- @ManyToOne - define a many - to - one relationship between the join tables
- @OneToMany - define a one - to - many relationship between the join tables
- @OneToOne - define a one - to - one relationship between the join tables

Entities annotation (3)

- Cascading – the action on the target entity will be applied to the associated entity
- javax.persistence.CascadeType
  - ALL
  - PERSIST
  - MERGE
  - REMOVE
  - REFRESH
  - DETACH

Entities annotation (3)

- Cascading – the action on the target entity will be applied to the associated entity
- javax.persistence.CascadeType
  - ALL
  - PERSIST
  - MERGE
  - REMOVE
  - REFRESH
  - DETACH

Hibernate

Hibernate (1)

- Hibernate is a Java framework that simplifies the development of Java application to interact with the database
- It is an open source, lightweight, ORM tool
- Implements the specifications of JPA for data persistence

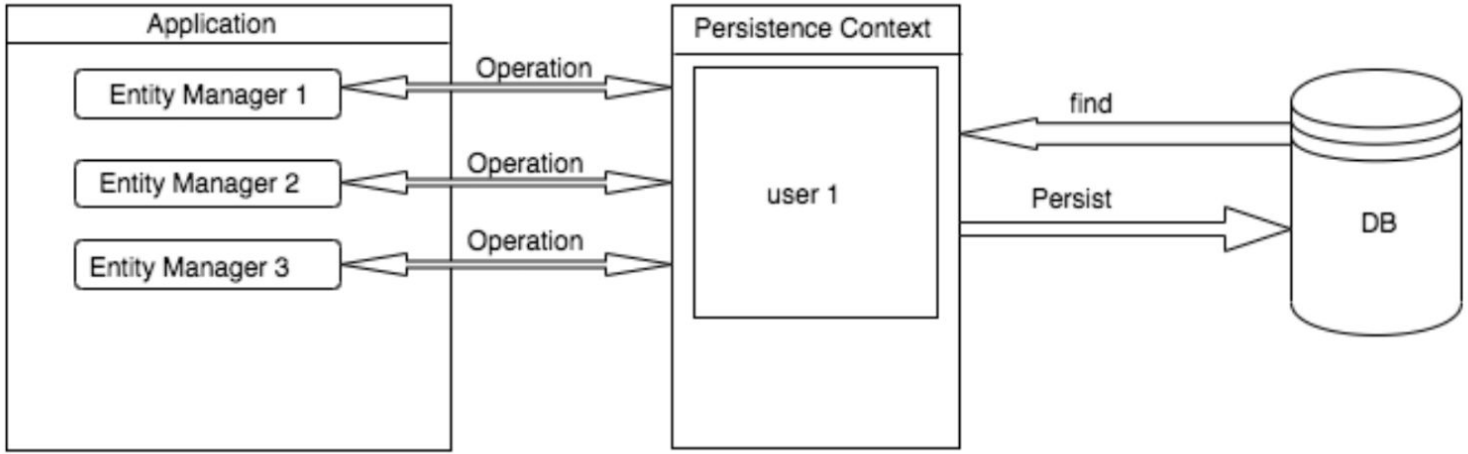
Hibernate (2)

- Advantages
  - Open Source and Lightweight
  - Fast – it uses 2 levels of cache
  - Database independent query – uses Hibernate Query Language, the object - oriented version of SQL
  - Automatic table creation
  - Simplifies complex joins

Persistence Context

- A persistence context is a set of entity instances in which for any persistent entity identity there is a unique entity instance
- Within the persistence context, the entity instances and their lifecycle are managed
- The EntityManager API is used to create and remove persistent entity instances, to find entities by their primary key, and to query over entities
- Persistence contexts are available in two types:
  - Transaction - scoped persistence context
  - Extended - scoped persistence context

Hibernate Overview



Transaction Persistence Context

Reference: https://www.baeldung.com/jpa-hibernate-persistence-context

Persistence context annotations

- @EnableTransactionManagement
  - Enables Spring's annotation - driven transaction management capability
  - Is added on the main class, that also has @SpringBootApplication

- @Transactional
  - Annotation that is added to all methods that change the database structure (insert, update, delete)

- @PersistenceContext
  - To inject the EntityManager

Example

```java
@Entity
@Table(name = "shopping_cart")
public class ShoppingCart {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private int id;

    @Column(name = "price", nullable = false)
    private float price;

    @OneToMany(cascade = CascadeType.PERSIST)
    @JoinColumn(name = "cart_id")
    private Set<ShoppingItem> shoppingItems;
```

ACID

Definition (1)

- In computer science, ACID (atomicity, consistency, isolation, durability) is a set of properties of database transactions intended to guarantee data validity despite errors, power failures, and other mishaps. I
- n the context of databases, a sequence of database operations that satisfies the ACID properties (which can be perceived as a single logical operation on the data) is called a transaction.

Definition (2)

- Atomicity
  - Atomicity guarantees that each transaction is treated as a single "unit", which either succeeds completely, or fails completely

- Consistency
  - Consistency ensures that a transaction can only bring the database from one valid state to another, maintaining database invariants: any data written to the database must be valid according to all defined rules, including constraints, cascades, triggers, and any combination thereof

Definition (3)

- Isolation
  - Transactions are often executed concurrently (e.g., multiple transactions reading and writing to a table at the same time)

- Durability
  - Durability guarantees that once a transaction has been committed, it will remain committed even in the case of a system failure (e.g., power outage or crash)

/ Q&A


OBILE/ACADEMY