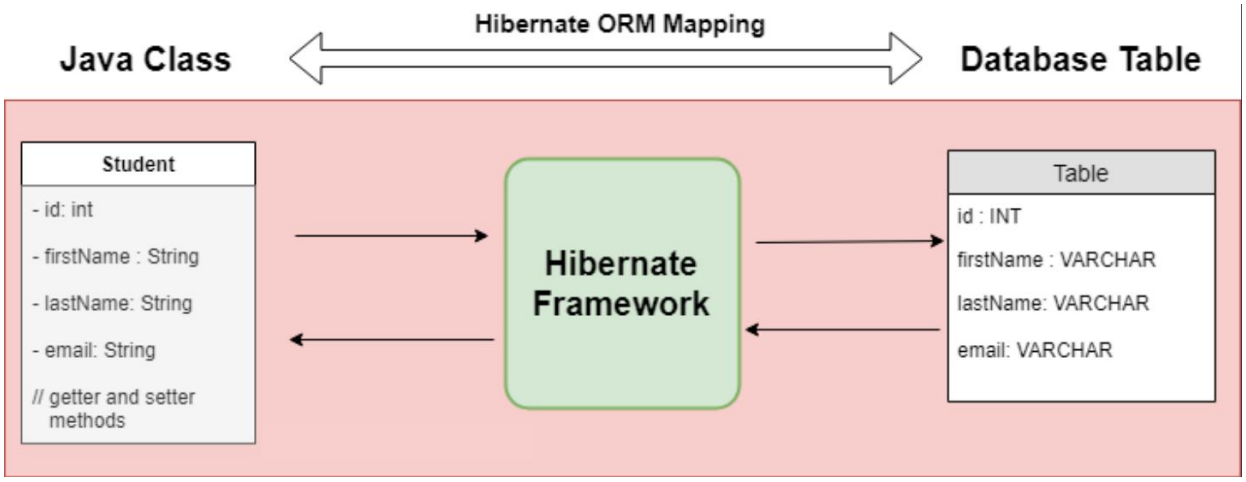
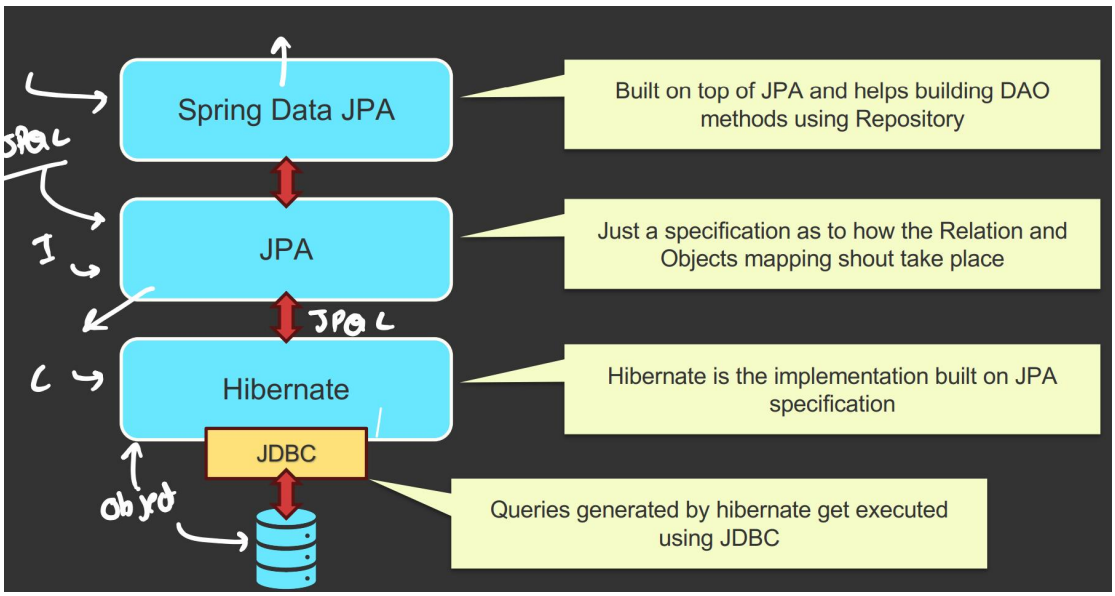
Hibernate ORM Mapping





- With the help of driver of particular database, we can connect JDBC to it.
  - Inside JDBC, we need to write SQL queries that will be supplied to Database.
- Then comes Hibernate, it is responsible for **Object-Relational-Mapping (ORM)**.
  - It'll convert the specific Java object to Database relational entities.
  - Hibernate is the implementation built on JPA specification.
- JPA is just a specification as to how the Relation and Objects mapping shout take place.
- Then it comes Spring JPA.
  - It is built on top of JPA and helps building DAO (Data Object) methods using Repository.
- We can write on top of Spring Data JPA that is high-level data manipulation methods. Or also we can write JPQL on JPA level.

The exact flow from Spring Data JPA to Database

- First Layer is **SPRING DATA JPA**
  - Built by Spring on top of **JPA**.
  - Contains:
    - **JpaRepository interface** (extended by *user-defined repository interfaces*).
    - **SimpleJpaRepository class** (contains *method bodies of JpaRepository*).
  - SimpleJpaRepository already has implementations of CRUD methods (defined inside EntityManager interface of JPA).
  - No method implementation injection at runtime.
  - Spring creates a proxy that forwards repository method calls to **SimpleJpaRepository**.
  - Dependencies like **EntityManager** are **injected** at runtime.
- 2nd Layer is **JPA**
  - Pure Java specification.
  - Defines annotations and interfaces like **EntityManager**.
  - **SimpleJpaRepository** calls methods of **EntityManager**.
  - JPA provides only contracts, no implementations.
- 3rd Layer is **JPA Provider** (**Hibernate** is mainly used)
  - **Hibernate** implements **EntityManager** interface.
  - Provides actual method definitions.
  - Generates SQL queries.
  - Passes SQL to JDBC.
- 4th Layer is **JDBC**
  - Java API for DB communication.
  - Executes SQL generated by Hibernate.
  - Sends SQL to database drivers.
- 5th Layer is **Database Driver**
  - Executes SQL on the database.
  - Performs actual DB operations.

- Hibernate
  - It is a powerful, high-performance Object-Relational-Mapping (ORM) framework that is widely used with Java.
  - It provides a framework for mapping an object-oriented domain model to a relational database.
  - It is one of the implementations of Java Persistence API (JPA) which is a standard specification for ORM in Java.
- JPA
  - It is a specification for ORM in Java.
  - It defines a set of interfaces and annotations for mapping Java objects to database tables and vice versa.
  - It itself is just a guideline, doesn't provide any implementations. Implementation is provided by JPA Provider framework like Hibernate.

Common Hibernate Configurations

- **spring.jpa.hibernate.ddl-auto=update/create/validate/create-drop/none** (1)
  - Update: we want to update the table when we update the entity
  - Create: everytime we running the server, old table will be dropped and create a new.
  - Validate: the table that we have and entity that we have are matching or not
  - Create-drop: create table on running of server and drop that after stopping the server (not used in production)
- **spring.jpa.show-sql=true** (2)
  - If we want to see all the queries being generated underneath
- **spring.jpa.properties.hibernate.format_sql=true** (3)
  - The queries coming from the previous command (2) should be displayed after properly beautifying not in a single line.
- **spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySql5Dialect** (optional) (4)
  - Defines the rule that hibernate will use to convert JPQL to queries.
  - Database are having their own dialect.
  - Its optional because it'll pick the proper dialect by itself.

- There are multiple annotations for **Entity** objects

```java
public class Product {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    // can't be nullable & max length = 23
    @Column(nullable = false, length = 20)
    private String sku;

    @Column(name = "title_x")
    private  String title;

    private BigDecimal price;

    private  Integer quantity;

    @CreationTimestamp
    private LocalDateTime createdAt;

    @UpdateTimestamp
    private LocalDateTime updatedAt;
}
```

  - **@Id**, **@GeneratedValue**, **@Column** (change name, nullable true or false, length if it's a string, etc etc), **@CreationTimeStamp**, **@UpdateTimeStamp** …etc

- **@Table** annotation

```java
@Table(
    name = "employees",
    catalog = "employee_catalog",
    schema = "hr",
    uniqueConstraints = {
        @UniqueConstraint(columnNames = {"email"})
    },
    indexes = {
        @Index(name = "idx_name", columnList = "name"),
        @Index(name = "idx_department", columnList = "department")
    }
)
```

  - There is something called **namespace** in database.
    - **auth.**user, **sales.**user
    - Here both have the same table name "user", but they do not conflict because they belong to different namespaces (auth, sales).

- In MySQL, the database acts as the namespace (mapped using **catalog** in **@Table**).
- In PostgreSQL and Oracle, the schema acts as the namespace (mapped using **schema** in **@Table**).
- So, schema and catalog both represent the same concept (**namespace**), and which one is used depends on the database.

```
UniqueConstraint[] uniqueConstraints() default {};
```

- UniqueConstraint is also an annotation :).
- `public @interface UniqueConstraint {`

```
@Table(
        name = "product_table",
        uniqueConstraints = {
            // column "sku" should be unique
            @UniqueConstraint(name = "sku_unique", columnNames = {"sku"}),
            // columns "title" & "price" combination should be unque
            @UniqueConstraint(name = "title_price_unique", columnNames = {"title_x", "price"})
        },
```

- (**title_x** because we have changed the column name to **title_x**; previous image)
- Name is used to provide a specific name to the constraint. Otherwise it'll generate some random unique name for the constraint.
- **name** is useful during debugging.
- `Duplicate entry 'a@b.com' for key 'UK_3ks8d9'` (without name)
- `Duplicate entry 'a@b.com' for key 'uk_user_email'` (with name)

indexes

- Here the **columnList** is a *String* not a *List*.
- You should give comma separated column names.

```
indexes = {
    @Index(name = "sku_index", columnList = "sku"),
    @Index(name = "title_price_index", columnList = "title, price")
}
```

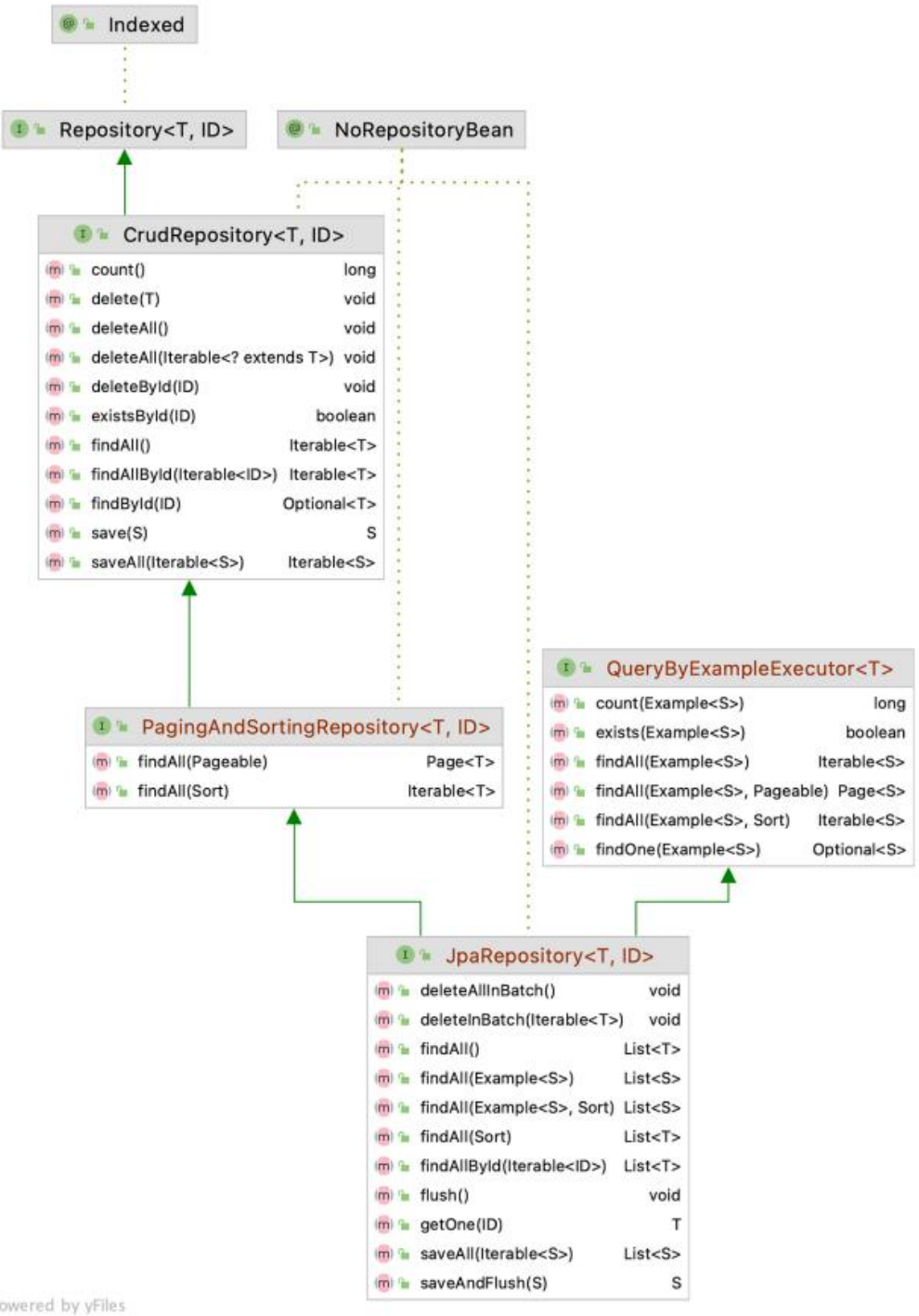
➢ NOTE: **database** should already be present. It'll not create the database inside the server by itself.

➢

- **Indexing** in database.
  - ```
    @Index(name = "idx_user_email", columnList = "email")
    ```
    (JPA)
  - ```
    CREATE INDEX idx_user_email ON users(email);
    ```
    (SQL)
  - An index is a *separate data structure* that stores indexed **column** values along with **row pointers**.
  - It is not a normal table; it is created and managed internally by the database..
    - ```
      a@x.com → row 5
      b@y.com → row 12
      ```
  - But this is not a normal table, it is created and managed by database itself.
  - **Read** queries are ***faster***, but **create**, **update**, **delete** queries are ***slower*** as it needs to update the index table as well.
  -
      ```

Spring Data JPA

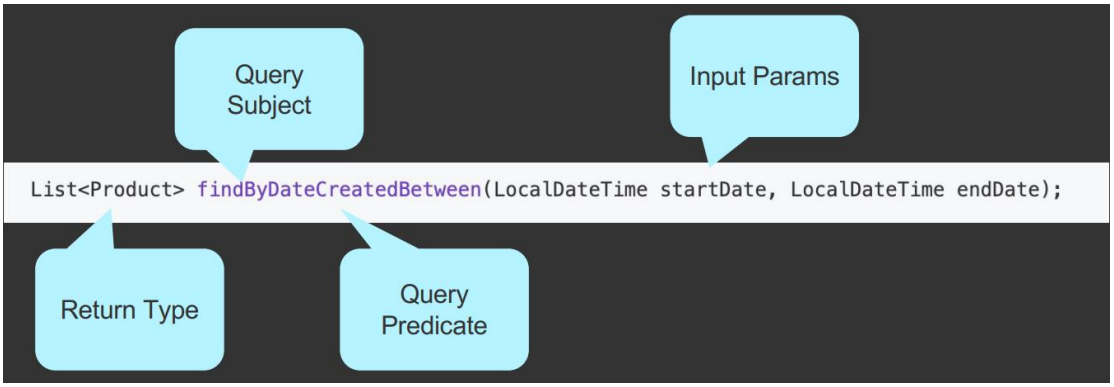
- It is a part of the larger Spring Data Family.
- It builds on top of JPA, providing a higher-level and more convenient abstraction for data access.
- Spring data JPA makes it easier to implement JPA-based repositories by providing boilerplate code, custom query methods, and various utilities to reduce the amount of code you need to write.



-

- **SimpleJpaRepository *class*** implements the *JpaRepository interface*. It contains implementation of all the methods of the JpaRepository and its parent interfaces.
- Key Features of Spring Data JPA
  - Repository Abstraction:
    - Provides a *Repository* interface with methods for common data access operations.
  - Custom Query Methods:
    - Allows defining custom query methods by simply declaring method names.
  - Pagination & Sorting:
    - Offers built-in support for pagination and sorting.
  - Query Derivation:
    - Automatically generates queries from method names.
- You'll have to just write the method name in the Repository and no need to implement. It'll be done automatically.
  -
```
@Repository  2 usages
public interface ProductRepository extends JpaRepository<ProductEntity, Long> {

    List<ProductEntity> findByTitle(String title);  1 usage
```
  - Just like this just write the method.
  - NOTE: If you remember the column name is **title_x** not **title**.
    -
```
@Column(name = "title_x")
private  String title;
```
    - **query generation takes place according to the Java object; not Database column**; If you write **findByTitleX** then it'll not work;

- **Rules for Creating Query Methods**
  - 

  - Return type will be mostly **Entity**, **Optional<Entity>** or **List<Entity>**
  - In the diagram, **Query Subject** is **findBy**, and **Query Predicate** is **DateCreatedBetween.**
  - The name of the query method must start with one of the following prefixes
    - find..By, read..By, query..By, get..By
    - Examples: **findByName, readByName, queryByName, getByName**
  - If we want to limit the number of returned query results, we can add the ***First*** or the ***Top*** keyword before the first by word**.**
    - Examples: **findFirstByName**, **readFirst2ByName**, **findTop10ByName**
  - If we want to select unique results, we have to add the Distinct keyword before the first ***By*** word.
    - Examples: **findDistinctByName or findNameDistinctBy** --- both are same
  - Combine property expression with ***And*** and ***Or***
    - Examples: **findByNameOrDescription, findByNameAndDescription**
  - For more, refer to the link: **query keyword reference**
- A few examples:
  - ```
    List<ProductEntity> findByCreatedAtAfter(LocalDateTime after);
    ```
    - To get all the items that were created after a particular time.
  - **findByQuantityGreaterThanAndPriceLessThan(int quantity, int price)**
    - The argument orders should be same as the query.
  -
  - F
  - F
  - f
    ```

- F
- F
- F
- F