Spring Breakout Session
Spring Data and JPA

by Vlad Costel Ungureanu
for FII UAIC,
Learn Stuff and Endava Pass It On

Spring Breakout Session

Spring Data and JPA

- Basic SQL
- Spring JDBC Template
- Helpers
- How to use it?
- JPA
- Data Source
- Persistence Unit
- Entity Manager
- Entities
- Entity Operations
- JPQL/HQL
- Parameters
- Results
- Named Queries
- Native SQL Queries
- ACID
- Transactions
- Overview

Basic SQL

```
CREATE TABLE table_name(
        id SERIAL UNIQUE NOT NULL,
        some_field filedType,
        PRIMARY KEY(id)
);
```

```
SELECT * FROM table_name WHERE condition GROUP BY condition HAVING condition ORDER
BY condition LIMIT 10
```

```
INSERT INTO table_name (1, “value”);
```

```
UPDATE table_name SET some_field = “some other value” WHERE id = 1;
```

```
DELETE FROM table_name WHERE id = 1;
```

Spring JDBC Template

```
<bean id='dataSource'
                class='org.springframework.jdbc.datasource.DriverManagerDataSource'>
            <property name='driverClassName' value='org.postgresql.Driver' />
            <property name='url' value='jdbc:postgresql://localhost:5432/skelet' />
            <property name='username' value='postgres' />
            <property name='password' value='postgres' />
</bean>

private JdbcTemplate jdbcTemplate;
private SimpleJdbcInsert insertSale;

@Autowired
public void setDataSource(DataSource dataSource) {
        this.jdbcTemplate = new JdbcTemplate(dataSource);
        this.insertSale = new SimpleJdbcInsert(dataSource).withTableName("SALE");
}
```

How to use it

❑ Insert:

```
Map<String, Object> parameters = new HashMap<String, Object>();
parameters.put(“some_field", “some_value”);
this.insertSale.execute(parameters);
```

❑ Select:

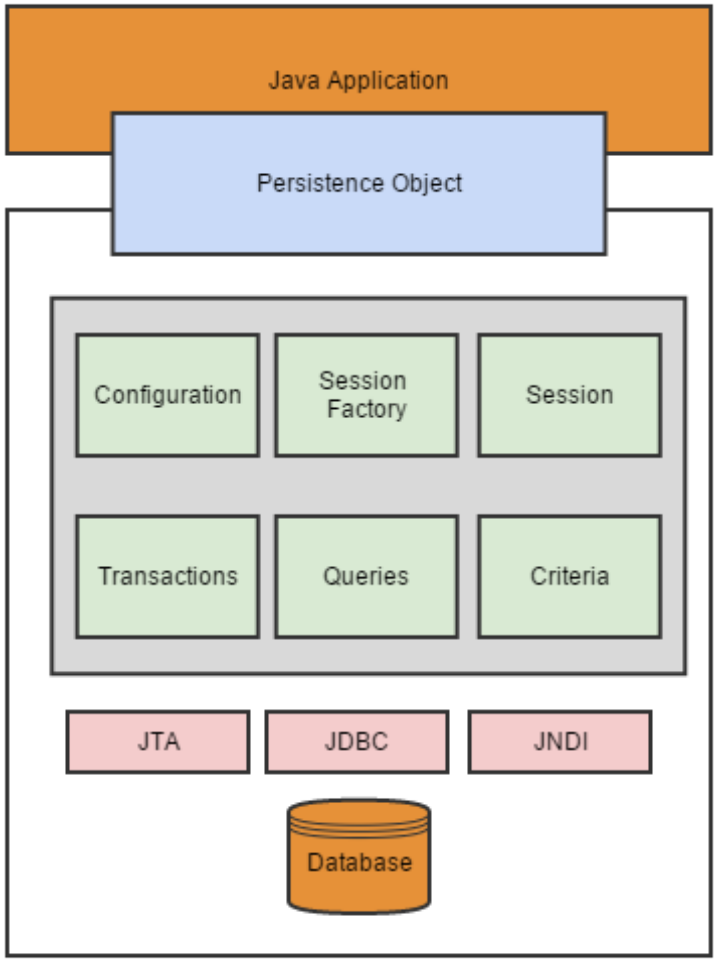
```
String sql = "SELECT * FROM CUSTOMER WHERE CUST_ID = ?";
Test test= (Test) jdbcTemplate().queryForObject(
                sql, new Object[] {id}, new TestRowMapper ());
```

Helpers

```
public class TestRowMapper implements RowMapper {

        @Override
        public Object mapRow(ResultSet rs, int line) throws SQLException {
                TestResultSetExtractor extractor = new TestResultSetExtractor();
                return extractor.extractData(rs);
        }
}
public class TestResultSetExtractor implements ResultSetExtractor<String>{

        @Override
        public Test extractData(ResultSet rs) throws SQLException,
                        DataAccessException {
                // read from rs to object
                return anObjectInstance;
        }
}
```


Java Application
Persistence Object
Configuration
Session Factory
Session
Transactions
Queries
Criteria
JTA
JDBC
JNDI
Database

Data Source

```
<data-source>
        <name>java:app/env/Application_Level_DataSource</name>
        <class-name>org.postgresql.ds.PGSimpleDataSource</class-name>
        <server-name>HOST</server-name>
        <port-number>PORT</port-number>
        <database-name>DATABASE_NAME</database-name>
        <user>USER</user>
        <password>PASSWORD</password>
        <property>
                <name>connectionAttributes</name>
                <value>;create=true</value>
        </property>
        <transactional>true</transactional>
</data-source>
```

Persistence Unit

persistence.xml:

```
<persistence xmlns="http://java.sun.com/xml/ns/persistence" version="1.0">
        <persistence-unit name=“jpaPoc" transaction-type="JTA">
                <jta-data-source>
                        java:app/env/Application_Level_DataSource
                </jta-data-source>
        </persistence-unit>
</persistence>
```

Entity manager

- **NEW**: entity is not persisted and it does not have and unique identifier within the persistence context
- **MANAGED**: entity is persisted and it has an unique identifier within the persistence context
- **DETACHED**: entity has an unique identifier but it is no longer associated with the persistence context
- **REMOVED**: entity has an unique identifier but is scheduled for deletion

Entity States – Id Generator

```
@Id
@GeneratedValue(strategy = “GenerationType.AUTO”)
//@GeneratedValue(strategy = “GenerationType.SEQUENCE”)
// many more possibilities
@Column(name = “id”)
private Long id;
```

Associations – One to Many

```
@Entity
public class Product {

@OneToMany
@JoinTable(name="PRODUCT_PARTS",
joinColumns=@JoinColumn(name="PRODUCT_ID",
inverseJoinColumns=@JoinColumn(name="PART_ID"))
   public Set<Part> parts;
}

@Entity
public class Part {
  ...
}
```

Associations – One to Many Bidirectional

```
@Entity
public class Product {

@OneToMany(cascade=CascadeType.ALL, mappedBy="product")
public Set<Part> parts;

}

@Entity
public class Part {

@ManyToOne(cascade=CascadeType.ALL)
private Product product;

}
```

Associations – Many to One

```
@Entity
public class Product {
 …
}


@Entity
public class Part {

@ManyToOne
@JoinColumn(name=“PRODUCT_ID ")
private Product product;
}
```

Associations – Many to One Join Table

```
@Entity
public class Product {
 …
}


@Entity
public class Part {

@ManyToOne
@JoinTable(name="PRODUCT_PARTS",
   joinColumns=@JoinColumn(name="PART_ID",
   inverseJoinColumns=@JoinColumn(name="PRODUCT_ID"))
private Product product;
}
```

Associations – One to one

```
@Entity
public class Body {

   @OneToOne
   @PrimaryKeyJoinColumn
   //@JoinColumn(name="heart_id")
   public Heart heart;
}

@Entity
public class Heart {
   @Id
   public Long id;
}
```

Entity operations

@PersistenceContext
private EntityManager em;

- Persist: em.persist(entity);
  - Inserts entity in DB if it does not have and identifier
  - Before commit(), entity manager will call flush()
  - Flush() will execute all operation held in cache
- Merge: em.merge(entity);
  - Modifies the state of the entity in the Database
  - Either copies the instance or creates a new one in the persistence context
  - Managed entities are ignored
  - Removed entities will throw exceptions
- Remove: em.remove(entity);
  - Deletes an entity from the DB
  - The actual time of deletion depends on the flush() call
  - Entities already scheduled for deletion will be ignored
  - Detached entities will throw exceptions
- Find: em.find(Class<T> entityClass, Object primaryKey);
  - Searches for the entity in DB based on the identifier
  - Returns the entity if found or null otherwise (might generate NullPointerException)


WHY ARE YOU
STILL AWAKE

JPQL/HQL

- Query q = em.createQuery("SELECT x FROM" + ENTITY_NAME+ " x");
- List< ENTITY_NAME > results = (List< ENTITY_NAME >) q.getResultList();
  - Queries support a similar sintax to SQL
  - query_statement ::= select_clause from_clause
    [where_clause][groupby_clause][having_clause][orderby_clause]
  - update_statement :: = update_clause [where_clause]
  - delete_statement :: = delete_clause [where_clause]

- **Where to use this?**
  - When the query depends on the user input
  - We don't know how the query will look until Runtime

- **Why not use this?**
  - JPQL is converted to SQL and validated
  - If we use String concatenation then the query will not be held in cache

Parameters

✓Query q = em.createQuery("SELECT x FROM ENTITY_NAME x WHERE x.field = :value“);

✓List< ENTITY_NAME > results =
(List< ENTITY_NAME >) q.setParameter(“value”, myValue).getResultList();

✓Query q = em.createQuery("SELECT x FROM ENTITY_NAME x WHERE x.field = ?1“);

✓List< ENTITY_NAME > results =
(List< ENTITY_NAME >) q.setParameter(1, myValue).getResultList();

Results

✓Query q = em.createQuery("SELECT x FROM ENTITY_NAME x“);

✓List< ENTITY_NAME > results =

- ✓ (List< ENTITY_NAME >) q.getResultList();
- ✓ (List< ENTITY_NAME >) q.getSingleResult();
- ✓ (List< ENTITY_NAME >) q.setMaxResults(myValue).getResultList();
- ✓ (List< ENTITY_NAME >) q.setLockMode(LockModeType*).getResultList();*
  - ✓ *Lock Mode Type:*
    - ✓ *Optimistic: presupune ca tu esti singurul care lucreaza cu data*
    - ✓ *Pessimistic: se asigura ca esti singurul care lucreaza cu data*

Named Queries

✓@NamedQuery(name="findAllEntities", query="SELECT x FROM ENTITY_NAME x
WHERE x.name LIKE :someName")

✓List< ENTITY_NAME > results = em.createNamedQuery(" findAllEntities")
.setParameter(" someName ", “my name").getResultList();

✓Where to use this ?
- ✓ Static queries
- ✓ Centralized declarations
- ✓ Validated at built time
- ✓ Held in cache at runtime, do not require any further valdiation

✓Why not use this ?
- ✓ Not customizable at Runtime

Native SQL

✓Query q = em.createNativeQuery("SELECT x FROM table_name x“, ENTITY_NAME.class);

✓List< ENTITY_NAME > results = (List< ENTITY_NAME >) q.getResultList();

✓When to use this?

- ✓ Fully use SQL advantages
- ✓ Can call stored procedures and functions
- ✓ Some things area easier to write in SQL
- ✓ No translation from JPQL/HQL to SQL

✓ **Why not use this?**

- ✓ Dependent on database and table structure
- ✓ Ignores all JPA/ORM advantages

Transactions - ACID

- **A**tomicity
- **C**onsistency
- **I**solation
- **D**urability

Transaction Management

✓Transactional annotations (@Transactional)

✓Propagation
- ✓ REQUIRED
- ✓ SUPPORTS
- ✓ MANDATORY
- ✓ REQUIRES_NEW
- ✓ NOT_SUPPORTED
- ✓ NEVER
- ✓ NESTED

Assignments

✓For the previous assignment add the Repository layer of the application

✓Use JPA with Hibernate for this assignment

✓Use criteria builder and enhance the search for the previous application

THANK YOU!

Vlad Costel Ungureanu
ungureanu_vlad_costel@yahoo.com

This is a free course from LearnStuff.io
– not for commercial use –