Please Complete The Socrative Pulse Survey

URL: gosocrative.com

Room Name: JAVAGOLD

Data Access Part 2

Objectives

- Housekeeping
- Review
- Exceptions in a DAO class
- INSERT, UPDATE, DELETE from Java
- Read data returned from an INSERT using RETURNING
- SQL Injection Attack

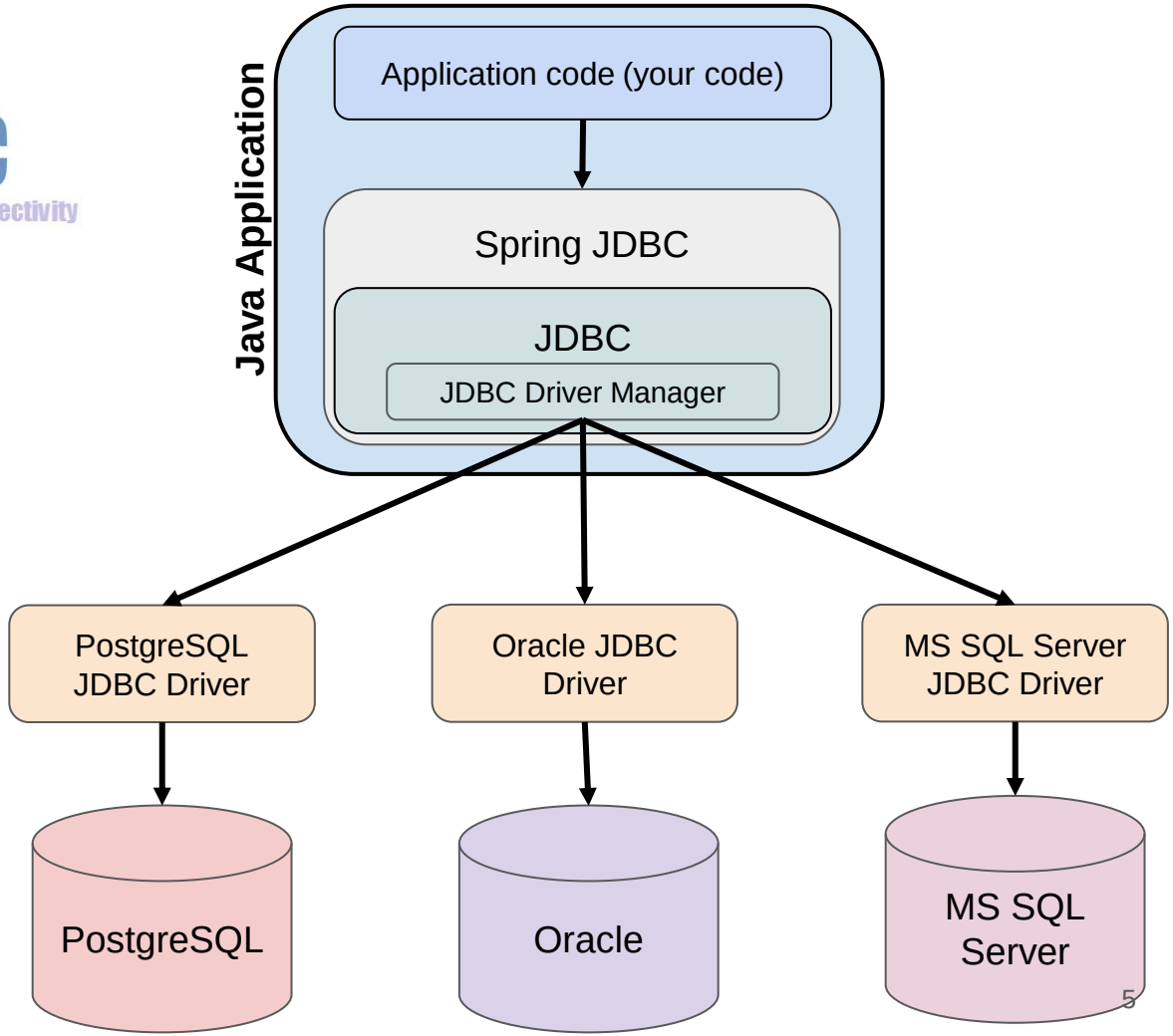
Housekeeping

- Due dates on exercises.
- Reading/tutorial/Quiz
- Daniel’s coding contest! Invite: www.hackerrank.com/nlr-13-coding-contest
- Try to remember your feedback on this week’s experimental curriculum.

SpringJDBC is a framework that provides an *abstraction for JDBC* making it easier to use.

Provides a consistent way to create queries, handle results, deal with exception, and automatically provides transactions.

Java Applications use the **SpringJDBC** Framework, which will utilize *JDBC* to call the *Vendor Supplied JDBC Driver* to access the *Database*.



DataSource

An object that represents a database.

Allows a Java application to connect to the database and use SQL commands.

```
BasicDataSource movieDBDataSource = new BasicDataSource();
dvdstoreDataSource.setUrl("jdbc:postgresql://localhost:5432/movieDB");
dvdstoreDataSource.setUsername("postgres");
dvdstoreDataSource.setPassword("postgres1");
```

`BasicDataSource` – The `org.apache.commons.dbcp2.BasicDataSource` provides the ability to make a database connection and creates a *Connection Pool*.

The **`setUrl`** takes a connection string as an argument

`setUsername()` and **`setPassword()`** methods set the username and password to use when connecting to the database

JDBCTemplate

org.springframework.jdbc.core.JdbcTemplate

- The central class of SpringJDBC
- Simplifies using JDBC and helps to avoid common errors.
- Allows execution of SQL Queries
- Provides a uniform way to retrieve results.

JDBCTemplate requires a **DataSource** as a constructor argument when instantiated.

```
JdbcTemplate jdbcTemplate = new JdbcTemplate(datasource);
```

JDBCTemplate Class

QueryForRowSet – performs query to the database

```
String sqlMoviesByReleaseYear = "SELECT * FROM movie WHERE release_date >= '01/01/2006' LIMIT 10";


SqlRowSet results = movieDBJdbcTemplate.queryForRowSet(sqlMoviesByReleaseYear);

System.out.println("Movies since 2006: ");
while(results.next()) {
        String movieTitle = results.getString("title");
        int releaseYr = results.getInt("release_year");
        System.out.println(movieTitle +" ("+ releaseYr +")");
}
```

SqlRowSet is a set containing all the data (rows) coming back from database

While loop loops through the results and turns the data being returned into Java data types to be displayed

SQL Parameters

```
String sqlMovieByReleaseYear = " SELECT * FROM film WHERE release_year >= " +
     movieReleaseYear + " LIMIT 10";
```

It is not a good idea to use the concatenation - better to use parameters

```
String sqlMovieByReleaseYear = " SELECT * FROM film WHERE release_year >= ? LIMIT 10";

int movieReleaseYear = 2006;
SqlRowSet results = movieDBJdbcTemplate.queryForRowSet(sqlMoviesByReleaseYear, movieReleaseYear);

System.out.println(movieReleaseYear +" Movies: ************************");
while(results.next()) {
   String movieTitle = results.getString("title");
   int releaseYr = results.getInt("release_year");
   System.out.println(movieTitle + " (" + releaseYr + ")");

}
```

DAO Pattern Step 1

- We start off with a Interface specifying that a class that chooses to implement the interface must implement methods to communicate with a database (i.e. search, update, delete). Consider the following example:

```
public interface CityDAO {  // CRUD – create, read, update, delete
     public void createCity(City city);  // c - create
     public City getCity(long cityId);  // r - read
}
```

DAO Pattern Step 2

```
public class JDBCCityDAO implements CityDAO {

     private JdbcTemplate jdbcTemplate;

     public JDBCCityDAO(DataSource dataSource) {
          this.jdbcTemplate = new JdbcTemplate(dataSource);
     }

     @Override
     public void createCity(City city) {
          String sqlInsertCity = "INSERT INTO city(id, name, countrycode, district, population) " +
                                 "VALUES(?, ?, ?, ?, ?)";
          newCity.setId(getNextCityId());
          jdbcTemplate.update(sqlInsertCity, city.getCityName(), city.getStateAbbreviation(),
                                 city.getPopulation(), city.getArea());
     }

     @Override
     public City  getCity(long id) {
         City theCity = null;
         String sqlFindCityById = "SELECT city_id, city_name, state_abbreviation, population, area "+
                                 "FROM city "+
                                 "WHERE city_id = ?";
         SqlRowSet results = jdbcTemplate.queryForRowSet(sqlFindCityById, id);
         if(results.next()) {
               theCity = mapRowToCity(results);
         }
         return theCity;
}
```

The contractual obligations of the interface are met.

DAO Pattern Step 3

- In our orchestrator class, we will be using a polymorphism pattern to declare our DAO objects:

```
dao.createCity(newCity);

City theCity = dao.getCity(id);
```

We can now call the methods that is defined in concrete class and required by the interface.

Exceptions in a DAO class

Exceptions with JDBC and DAO

- Exceptions are when an unexpected error occurs.
  - (FileNotFoundException – checked exception for file handling)

- Exceptions that can occur with DAO methods and JDBC
  - CannotGetJdbcConnectionException
  - (Custom Exceptions)
  - BadSqlGrammarException
  - DataIntegrityViolationException

CannotGetJdbcConnectionException

- Unable to connect to the database or to a server
- Can occur with any CRUD operation

```
[INFO] 2020-12-03 22:44:26.913  - [taskAppId=TASK-360-52582-96413]:[499] - sql Params are replaced sql , parameters:
[INFO] 2020-12-03 22:44:26.913  - [taskAppId=TASK-360-52582-96413]:[52] - can't find udf function resource
[INFO] 2020-12-03 22:44:26.922  - [taskAppId=TASK-360-52582-96413]:[413] - prepare statement replace sql : com.mysql.jdbc.JDBC4PreparedStatement@7cf992
[ERROR] 2020-12-03 22:46:26.928  - [taskAppId=TASK-360-52582-96413]:[242] - execute sql error
org.mybatis.spring.MyBatisSystemException: nested exception is org.apache.ibatis.exceptions.PersistenceException:
### Error querying database.  Cause: org.springframework.jdbc.CannotGetJdbcConnectionException: Failed to obtain JDBC Connection; nested exception is c
eption: wait millis 60000, active 0, maxActive 50, creating 1, createElapseMillis 80093, createErrorCount 8
### The error may exist in file [/opt/ds-1.3.2-agent/conf/org/apache/dolphinscheduler/dao/mapper/UserAlertGroupMapper.xml]
### The error may involve org.apache.dolphinscheduler.dao.mapper.UserAlertGroupMapper.listUserByAlertgroupId
### The error occurred while executing a query
### Cause: org.springframework.jdbc.CannotGetJdbcConnectionException: Failed to obtain JDBC Connection; nested exception is com.alibaba.druid.pool.GetC
 active 0, maxActive 50, creating 1, createElapseMillis 80093, createErrorCount 8
        at org.mybatis.spring.MyBatisExceptionTranslator.translateExceptionIfPossible(MyBatisExceptionTranslator.java:78)
        at org.mybatis.spring.SqlSessionTemplate$SqlSessionInterceptor.invoke(SqlSessionTemplate.java:440)
        at com.sun.proxy.$Proxy91.selectList(Unknown Source)
        at org.mybatis.spring.SqlSessionTemplate.selectList(SqlSessionTemplate.java:223)
        at com.baomidou.mybatisplus.core.override.MybatisMapperMethod.executeForMany(MybatisMapperMethod.java:158)
        at com.baomidou.mybatisplus.core.override.MybatisMapperMethod.execute(MybatisMapperMethod.java:76)
```

BadSqlGrammarException

- Error in the Sql statement in Java
- Can occur with any CRUD operation

```
[ERROR] 2023-02-06 14:47:37.145 [8888] org.apache.dolphinscheduler.api.exceptions.ApiExceptionHandler:[53] - 运行工作流实例错误
org.springframework.jdbc.BadSqlGrammarException:
### Error updating database.  Cause: org.postgresql.util.PSQLException: ERROR: syntax error at or near "DUPLICATE"
  位置: 145
### The error may exist in file [E:\idea\openSources\dolphinscheduler_xxj\dolphinscheduler-dao\target\classes\org\apache\dolphinscheduler\dao\mapper\TriggerRelationMapper.xml]
### The error may involve defaultParameterMap
### The error occurred while setting parameters
### SQL: INSERT INTO t_ds_trigger_relation (trigger_code, trigger_type, job_id, create_time, update_time) VALUES(            ?,?,?,?,?)         ON DUPLICATE KEY UPDATE update_time = ?;
### Cause: org.postgresql.util.PSQLException: ERROR: syntax error at or near "DUPLICATE"
  位置: 145
; bad SQL grammar []; nested exception is org.postgresql.util.PSQLException: ERROR: syntax error at or near "DUPLICATE"
  位置: 145
    at org.springframework.jdbc.support.SQLErrorCodeSQLExceptionTranslator.doTranslate(SQLErrorCodeSQLExceptionTranslator.java:239)
    at org.springframework.jdbc.support.AbstractFallbackSQLExceptionTranslator.translate(AbstractFallbackSQLExceptionTranslator.java:70)
    at org.mybatis.spring.MyBatisExceptionTranslator.translateExceptionIfPossible(MyBatisExceptionTranslator.java:91)
    at org.mybatis.spring.SqlSessionTemplate$SqlSessionInterceptor.invoke(SqlSessionTemplate.java:441) <1 internal line>
    at org.mybatis.spring.SqlSessionTemplate.insert(SqlSessionTemplate.java:272)
    at com.baomidou.mybatisplus.core.override.MybatisMapperMethod.execute(MybatisMapperMethod.java:59)
```

DataIntegrityViolationException

- Violating a constraint
  - Invalid primary key, foreign key, violating a unique constraint
- Can occur on INSERT and UPDATE statements in Java

```
throwable = {DbActionExecutionException@6855} "org.springframework.data.relational.core.conversion.DbActionExecutionException: Failed to execute DbAction.InsertRoot(entity=c... View
detailMessage = "Failed to execute DbAction.InsertRoot(entity=com.gerps.UserServices.domain.partybc.party.Party@30dcb029)"
cause = {DataIntegrityViolationException@6861} "org.springframework.dao.DataIntegrityViolationException: PreparedStatementCallback; SQL [INSERT INTO "party" ("currency_uor... View
stackTrace = {StackTraceElement[97]@6865}
suppressedExceptions = {Collections$UnmodifiabI

org.springframework.dao.DataIntegrityViolationException: PreparedStatementCallback;
SQL [INSERT INTO "party" ("currency_uom_id", "description", "party_id",
"party_type_id", "status_id", "version") VALUES (?, ?, ?, ?, ?, ?)]; ERROR: null
value in column "party_id" violates not-null constraint
 Detail: Failing row contains (null, null, Organisation, USD, Enable, 1).; nested
exception is org.postgresql.util.PSQLException: ERROR: null value in column
"party_id" violates not-null constraint
 Detail: Failing row contains (null, null, Organisation, USD, Enable, 1).
```

Custom Exceptions

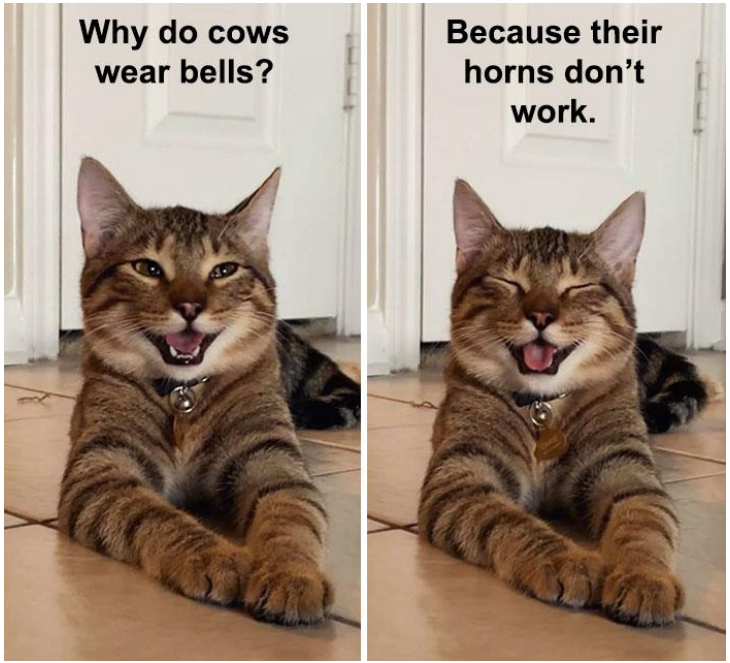
- Discussed last module that we can create our own custom exceptions
  - Extend RuntimeException

```java
// DaoException.java

package com.bookstore.exception;

public class DaoException extends RuntimeException {
    public DaoException() {
        super();
    }
    public DaoException(String message) {
        super(message);
    }
    public DaoException(String message, Exception cause) {
        super(message, cause);
    }
}
```

Let’s look at the code!



INSERT data into a database with no return data

- Two methods that can be used with INSERT statements
  - Update
  - queryForRowObject

- For UPDATE, INSERT, and DELETE statements we use the **.update** method instead of the .queryForRowSet method.

```
SqlRowSet results = jdbcTemplate.update(sqlString);
// Where sqlString contains an UPDATE, INSERT, or DELETE.
```

Executing SQL Statements that return a single result

Some Queries return a single piece of data.

A common usage is using RETURNING with an insert statement to have the generated id (sequence) returned. queryForObject() can be used to retrieve a single value.

```
String sql = "INSERT INTO city(name, countrycode, district,
population) VALUES(?, ?, ?, ?) RETURNING id"

Integer cityId = jdbcTemplate.queryForObject(sql, Integer.class,
                                "Smallville", "USA", "Kansas", 45001);
```

Executing SQL Statements that return a single result

Because the RETURNING will be sending back an integer, the queryForObject must specify the data type of the object being sent back.

Since id is an Integer, we use Integer.class

```
String sql = "INSERT INTO city(name, countrycode, district,
population) VALUES(?, ?, ?, ?) RETURNING id"

int cityId = jdbcTemplate.queryForObject(sql, Integer.class,
                          "Smallville", "USA", "Kansas", 45001);
```