Oracle JDBC has to be downloaded for oracle site not available in maven remote repository. But in may company have there own maven central NEXCUS

<repository>

<id>att\_dcae</id>

<name>att\_dcae</name>

<url>http://mavencentral.it.att.com:8081/nexus/repository/att-repository-snapshots</url>

</repository>

We can download directly.

<https://mkyong.com/spring-boot/spring-boot-spring-data-jpa-oracle-example/>

!-- Spring data JPA, default tomcat pool, exclude it -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

<exclusions>

<exclusion>

<groupId>org.apache.tomcat</groupId>

<artifactId>tomcat-jdbc</artifactId>

</exclusion>

</exclusions>

</dependency>

<!-- Oracle JDBC driver -->

<dependency>

<groupId>com.oracle</groupId>

<artifactId>ojdbc7</artifactId>

<version>12.1.0</version>

</dependency>

# create and drop tables and sequences, loads import.sql

spring.jpa.hibernate.ddl-auto=create-drop

# Oracle settings

spring.datasource.url=jdbc:oracle:thin:@localhost:1521:xe

spring.datasource.username=system

spring.datasource.password=password

spring.datasource.driver-class-oracle.jdbc.driver.OracleDriver

# logging

logging.pattern.console=%d{yyyy-MM-dd HH:mm:ss} %-5level %logger{36} - %msg%n

logging.level.org.hibernate.SQL=debug

#logging.level.org.hibernate.type.descriptor.sql=trace

logging.level.=error

If import.sql is found in the classpath, Hibernate will load it automatically.

Reset all jap concept.

**Stored procedure**

A **stored procedure** is a prepared SQL code that you can save, so the code can be reused over and over again. So if you have an SQL query that you write over and over again, save it as a **stored procedure**, and then just call it to execute it.

create or replace PROCEDURE Process\_GCP\_TEK\_TVSP\_Feed\_MIG(

p\_rcdType IN VARCHAR2,

p\_NODETYPE IN VARCHAR2,

p\_NODENAME IN VARCHAR2,

p\_IP\_ADDRS IN VARCHAR2,

p\_return OUT VARCHAR2

)

IS

v\_procName VARCHAR2(30 byte) := 'Process\_GCP\_TEK\_TVSP\_Feed\_MIG';

v\_notFound BOOLEAN := FALSE;

vTVSPNodeRec TVSP\_VOIPNODES%ROWTYPE;

v\_log VARCHAR2(1000);

v\_where\_key VARCHAR2(2000);

v\_updSql VARCHAR2(1000);

BEGIN

v\_where\_key := ' WHERE NodeName =''' || p\_NODENAME || ''' AND NodeType=''' || p\_NODETYPE ||

''' AND valid\_to is null'; -- AND ip\_addrs is not null

v\_updSql:= 'UPDATE tvsp\_voipnodes SET IP\_ADDRS=''' || p\_IP\_ADDRS || '''' || v\_where\_key;

Begin

SELECT \* INTO vTVSPNodeRec FROM TVSP\_VOIPNODES

WHERE NodeName = p\_NODENAME AND NodeType = p\_NODETYPE

AND valid\_to is null -- AND IP\_ADDRS is not null

FOR UPDATE NOWAIT;

-- v\_getTVSPNodeSql:= 'SELECT \* INTO vTVSPNodeRec FROM TVSP\_VOIPNODES'

--|| v\_where\_key || ' FOR UPDATE NOWAIT'; not work

-- execute immediate v\_getTVSPNodeSql;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

v\_notFound := TRUE;

WHEN TOO\_MANY\_ROWS THEN

p\_return := 'ERR ! 2+ same TGs found in TVSP\_VOIPNODES table';

-- raise\_application\_error(SQLCODE, SQLERRM);

WHEN OTHERS THEN

v\_log := 'Err=' ||SQLCODE || ' during find match= '|| SUBSTR(SQLERRM,1,500)

|| v\_where\_key;

p\_return := v\_log;

--raise\_application\_error(SQLCODE, SQLERRM);sa

End;

IF (UPPER(p\_rcdType) = 'M')

THEN

IF ((vTVSPNodeRec.IP\_ADDRS is null AND p\_IP\_ADDRS is not null) OR vTVSPNodeRec.IP\_ADDRS != p\_IP\_ADDRS)

THEN

execute immediate v\_updSql;

commit;

p\_return := 'SUCCESS';

ELSE

p\_return := 'Didnt Match';

END IF;

END IF;

END Process\_GCP\_TEK\_TVSP\_Feed\_MIG;

CallableStatement stmt = **null**;

Connection conn = **null**;

conn.setAutoCommit(**false**);

stmt = conn.prepareCall("{call Process\_GCP\_TEK\_TVSP\_Feed\_MIG(?,?,?,?,?)}");

stmt.setString(1, checkString(dataMap.get("p\_rcdType")));

stmt.setString(2, checkString(dataMap.get("p\_NODETYPE")));

stmt.setString(3, checkString(dataMap.get("p\_NODENAME")));

stmt.setString(4, checkString(dataMap.get("p\_IP\_ADDRS")));

stmt.registerOutParameter(5, Types.***VARCHAR***);

*logger*.info("before excuateStoredProcMigration: Process\_GCP\_TEK\_TVSP\_Feed\_MIG return = " + retStr);

stmt.execute();

*logger*.info("after excuateStoredProcMigration");

retStr = stmt.getString(5);

*logger*.info("after excuateStoredProcMigration: Process\_GCP\_TEK\_TVSP\_Feed\_MIG return = " + retStr);

conn.commit();

stmt.close();

conn.close();

Spring Boot way:

In summary:

* For Stored Procedure, SimpleJdbcCall.execute.
* For Stored Function, SimpleJdbcCall.executeFunction

import org.springframework.jdbc.core.JdbcTemplate;

import org.springframework.jdbc.core.namedparam.MapSqlParameterSource;

import org.springframework.jdbc.core.namedparam.SqlParameterSource;

import org.springframework.jdbc.core.simple.SimpleJdbcCall;

@Component

public class StoredProcedure1 {

@Autowired

private JdbcTemplate jdbcTemplate;

private SimpleJdbcCall simpleJdbcCall;

@PostConstruct

void init() {

// o\_name and O\_NAME, same

jdbcTemplate.setResultsMapCaseInsensitive(true);

simpleJdbcCall = new SimpleJdbcCall(jdbcTemplate)

.withProcedureName("get\_book\_by\_id");

}

Optional<Book> findById(Long id) {

SqlParameterSource in = new MapSqlParameterSource()

.addValue("p\_id", id);

Optional result = Optional.empty();

try {

Map out = simpleJdbcCall.execute(in);

if (out != null) {

Book book = new Book();

book.setId(id);

book.setName((String) out.get("O\_NAME"));

book.setPrice((BigDecimal) out.get("O\_PRICE"));

result = Optional.of(book);

}

} catch (Exception e) {

// ORA-01403: no data found, or any java.sql.SQLException

System.err.println(e.getMessage());

}

return result;

}

}