WORKSHOP SQL

Contents

[Project setup 2](#_Toc417658886)

[Data Definition Language 2](#_Toc417658887)

[Constraints 3](#_Toc417658888)

[Data Manipulation Language 4](#_Toc417658889)

[Retrieving Data from database 4](#_Toc417658890)

[Views 5](#_Toc417658891)

[Single row functions and group functions 5](#_Toc417658892)

[Liquibase generation DDLs 6](#_Toc417658893)

# Project setup

1. Let’s start our database using Docker:
   1. Download and install Docker Toolbox. Please make sure you have enabled virtualization from your BIOS.
   2. Run Docker Quickstart Terminal **as Administrator**.
   3. If you have not downloaded the Oracle XE image using Docker, do it using the following command: **docker pull wnameless/oracle-xe-11g**
   4. Run your Docker image with Oracle XE using the following command: **docker run -d -p 49160:22 -p 49161:1521 -e ORACLE\_ALLOW\_REMOTE=true wnameless/oracle-xe-11g**
   5. Your virtual machine will be available on the default IP provided by Docker. Usually, it’s 192.168.99.104 or something similar, but you can always see yours in the Docker console. The port will be 49161, forwarded from the port 1521 on the virtual machine (the database connection port)
2. If you have a functional Maven project from your previous presentation, use that one. Otherwise, open Intellij IDEA -> File -> New Project -> Maven Project -> See the lines below -> Next -> Set project name and location -> Finish

*GroupId: ro.teamnet.zth*

*ArtifactId: ZTH*

*Version: 1.0-SNAPSHOT*

1. Click on Database -> click on “+” icon -> Data Source -> Oracle and set the following -> OK
   1. Host: 192.168.99.104
   2. Port: 49161
   3. Database(SID): xe
   4. User: system
   5. Password: oracle
2. Let’s create our own database user:
   1. Connect to the database with the previously created configuration.
   2. Run the scripts in *create\_user.sql,* after carefully reading the comments and modifying the scripts accordingly.
   3. Modify the database connection in order to connect to your user, and then reconnect to it.

# Data Definition Language

1. Create a table LOCATIONS with the following columns and data types:

LOCATION\_ID NUMBER PRIMARY KEY,

STREET\_ADDRESS VARCHAR2(40),

POSTAL\_CODE VARCHAR2(12),

CITY VARCHAR2(30) NOT NULL,

STATE\_PROVINCE VARCHAR2(25)

1. Create a table DEPARTMENTS with the following columns and data types:

DEPARTMENT\_ID NUMBER PRIMARY KEY,

DEPARTMENT\_NAME VARCHAR2(30) NOT NULL,

LOCATION\_ID NUMBER

1. Create a table JOBS with the following columns and data types:

JOB\_ID NUMBER PRIMARY KEY,

JOB\_TITLE VARCHAR2(35) NOT NULL,

MIN\_SALARY NUMBER(6),

MAX\_SALARY NUMBER(6)

1. Create a table EMPLOYEES with the following columns and data types:

EMPLOYEE\_ID NUMBER PRIMARY KEY,

FIRST\_NAME VARCHAR2(20),

LAST\_NAME VARCHAR2(25) NOT NULL,

EMAIL VARCHAR2(25) NOT NULL,

PHONE\_NUMBER VARCHAR2(20),

HIRE\_DATE DATE NOT NULL,

JOB\_ID NUMBER NOT NULL,

SALARY NUMBER(8,2),

COMMISSION\_PCT NUMBER(2,2),

MANAGER\_ID NUMBER,

DEPARTMENT\_ID NUMBER

1. Create 4 sequences like below:

CREATE SEQUENCE TAB\_DEPARTMENTS\_SEQ

START WITH 1

INCREMENT BY 1;

CREATE SEQUENCE TAB\_EMPLOYEES\_SEQ

START WITH 1

INCREMENT BY 1;

CREATE SEQUENCE TAB\_JOBS\_SEQ

START WITH 1

INCREMENT BY 1;

CREATE SEQUENCE ZTH\_SEQ

START WITH 406

INCREMENT BY 1;

# Constraints

1. Create relations between the tables by altering the tables and adding the FOREIGN KEYS:

FK\_EMPLOYEES\_DEPARTMENTS – every employee should have a department

FK\_EMPLOYEES\_JOBS – every employee should have a job

FK\_EMPLOYEES\_EMPL\_MANAGER – every employee should have a manager, who is also an employee

SYNTAX:

ALTER TABLE <TABLE1> ADD CONSTRAINT <CONSTRAINT\_NAME> FOREIGN KEY (<FIELD\_TABLE1>) REFERENCES <TABLE2> (<FIELD\_TABLE2>);

# Data Manipulation Language

1. Insert example:

Insert a new record in the tables DEPARTMENTS and EMPLOYEES:

insert into departments values

( TAB\_DEPARTMENTS\_SEQ.nextval,

'Administration',

1700

);

insert into jobs values

(TAB\_JOBS\_SEQ.nextval

, 'President'

, 20000

, 40000

);

INSERT INTO employees

VALUES (TAB\_EMPLOYEES\_SEQ.nextval,

'Steven',

'King',

'SKING',

'515.123.4567',

sysdate,

'AD\_PRES',--value from second insert

24000,

NULL,

NULL,

90);--value from first insert

1. Run a **ROLLBACK** command.
2. Delete example – clean up the database. Run delete statements on all the tables we created:

DELETE FROM departments;

DELETE FROM locations;

DELETE FROM jobs;

DELETE FROM employees;

You can also use the TRUNCATE keyword.

It’s very important to delete data in the order of the references (foreign keys). For example, since a department has a location\_id, you cannot delete the location before you delete the department because it is referenced by the department.

1. Insert exercises – Run the scripts from *import\_values\_locations.sql* and *import\_values.sql*.
2. Run a **COMMIT** command.

# Retrieving Data from database

1. Simple SELECT queries:
2. Retrieve all employees from the database.
3. Retrieve all departments from the database.
4. Retrieve all jobs from the database.
5. Retrieve only the first name and last name of all employees.
6. Retrieve all employees from department 50
7. Increase salary by 30% for all employees in department 50 (UPDATE Statement)
8. Remove the employee with EMPLOYEE\_ID 101 (DELETE Statement)
9. Find all employees with job IT\_PROG, ordered by their first name.
10. Use an alias in a query:

select count(employee\_id) from employees emp where emp.JOB\_ID = 'IT\_PROG'

1. Same as 2, but also add the department name. (JOIN Statement)
2. Find all employees from Seattle (location\_id = 1700);

select 1.3\*salary from employees where department\_id=50;

update employees set manager\_id=102 where manager\_id=101;

delete from employees where manager\_id=101;

select \* from employees where job\_id='IT\_PROG' order by first\_name;

select count(employee\_id) from employees emp where emp.JOB\_ID = 'IT\_PROG';

select e.department\_id employees e, departments d where e.departments\_id=50 and d.departments\_id = e.department\_id;

# Views

1. Create a View with employee\_id, first\_name and department\_name.

# Single row functions and group functions

1. Run the following SQL: SELECT sysdate from dual;
2. Format the system date with **TO\_CHAR** function.

select to\_char(sysdate, 'dd-MM-yyyy') from dual

1. Convert and show a date from a varchar2 field

select to\_date ('25-11-2014', 'dd-MM-yyyy') from dual

1. Find all employees First Name, with Upper case and Email with Lower case
2. Find all employees First Name with the prefix “First Name: ” using || operator
3. Use the COUNT function: calculate the total number of employees.
4. Calculate and show the number of employees with job IT\_PROG
5. Calculate and show the number of employees for each department (count + group by)
6. Calculate and show the average salary for all employees in department 50 (use the AVG function)
7. Calculate and show max, min salary from employees who work in Seattle.

# Liquibase generation DDLs

1. Get the JDBC driver for Oracle from the Git repository
2. Open a command prompt as administrator, go in the directory where the ojdbc6.jar stands and run the following command:

mvn install:install-file -DgroupId=com.oracle -DartifactId=ojdbc6 -Dversion=11.2.0 -Dpackaging=jar -Dfile=ojdbc6.jar -DgeneratePom=true

1. Add the Oracle driver dependency inside the pom.xml:

<dependency>

<groupId>com.oracle</groupId>

<artifactId>ojdbc6</artifactId>

<version>11.2.0</version>

</dependency>

1. Add liquibase plugin in pom.xml:

<plugins>

<plugin>

<groupId>org.liquibase</groupId>

<artifactId>liquibase-maven-plugin</artifactId>

<version>3.0.5</version>

<configuration>

<changeLogFile> src/main/resources/db/changelog/db.changelog-master.xml </changeLogFile>

<driver>oracle.jdbc.driver.OracleDriver</driver>

<url>jdbc:oracle:thin:@ 192.168.99.104:49161:xe</url>

<username>username</username>

<password>password</password>

</configuration>

<executions>

<execution>

<phase>process-resources</phase>

<goals>

<goal>update</goal>

</goals>

</execution>

</executions>

</plugin>

</plugins>

1. Modify username and password tag from in the above plugin configuration
2. Create a *db* folder in resources; in *db* folder create a *changelog* folder
3. In *changelog* folder create 2 files: *db.changelog-master.xml and db.changelog-1.0.xml* with the following structure:

<?xml version="1.0" encoding="UTF-8"?>

<databaseChangeLog

xmlns="http://www.liquibase.org/xml/ns/dbchangelog/1.9"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.liquibase.org/xml/ns/dbchangelog/1.9

<http://www.liquibase.org/xml/ns/dbchangelog/dbchangelog-1.9.xsd>">

</databaseChangeLog>

1. In  *db.changelog-1.0.xml* create a *changeSet* for creating a table *PEOPLE* with the following fields:

PERS\_ID NUMBER(6) PRIMARY KEY,

FIRST\_NAME VARCHAR2(40 BYTE),

LAST\_NAME VARCHAR2(40 BYTE),

EMAIL VARCHAR2(30 BYTE) NOT NULL,

PHONE\_NUMBER VARCHAR2(25 BYTE),

ADDRESS\_ID NUMBER(6) NOT NULL,

1. Include *db.changelog-1.0.xml* in *db.changelog-master.xml*
2. Run the compile task in Maven
3. Create *db.changelog-2.0.xml* in which you will create a changeset with ADDRESS table which will have the following fields:

ADR\_ID NUMBER PRIMARY KEY,

STREET VARCHAR2(50 BYTE),

NO NUMBER(6),

CITY VARCHAR2(30 BYTE) NOT NULL,

1. In same file *db.changelog-2.0.xml* create another change set with a constraint for ADDRESS\_ID from the *PEOPLE* table
2. Include *db.changelog-2.0.xml* in *db.changelog-master.xml*
3. Run the compile task in Maven