

RIGA TECHNICAL UNIVERSITY

FACULTY OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY

INSTITUTE OF APPLIED COMPUTER SYSTEMS

“Technology of Large Databases”

Practical assignment #1

**Object-relational DB storage structures**

Author: Doston hamrakulov

Studentcardno.: 151ADB089

2016 / 2017study year

Contents

[1. Assignment description: 3](#_Toc499401609)

[2. Java class creation and loading in database: 4](#_Toc499401610)

[2.1 Using LOADJAVA program 4](#_Toc499401611)

[2.1.1 Define a class, Student, as follows: 4](#_Toc499401612)

[2.1.2 Compile the class on your client system using the standard Java compiler, as follows 5](#_Toc499401613)

[2.1.3 Load the class on the server using loadjava. You must specify the user name and password. Run the loadjava command as follows: 5](#_Toc499401614)

[2.1.4 Publish the stored procedure through a call specification 5](#_Toc499401615)

[2.1.5 Call the stored procedure, as follows: 6](#_Toc499401616)

[2.2 Using CREATE JAVA command. 6](#_Toc499401617)

[2.2.1 Create Java a file named Billionaire in SQL Developer using CREATE JAVA 6](#_Toc499401618)

[2.2.2 Publish the stored procedure through a call specification 7](#_Toc499401619)

[2.1.3 Call the stored procedure, as follows: 7](#_Toc499401620)

[2.3 Metadata 8](#_Toc499401621)

[3. Java Class meta-data inspection (Data input) and class methods of execution SQL command (data output). 9](#_Toc499401622)

[3.1 About planning database schema 9](#_Toc499401623)

[3.3 Writing the Java classes 11](#_Toc499401624)

[3.4 Loading the Java classes 14](#_Toc499401625)

[3.5 Publishing the Java classes 14](#_Toc499401626)

[3.6 Calling the Java stored Procedure 16](#_Toc499401627)

[4 Defining methods with the Java class and its application 19](#_Toc499401628)

[5 Conclusions. 19](#_Toc499401629)

# Assignment description:

1. Java class creation and loading in database:
   1. Using LOADJAVA program.
   2. Using CREATE JAVA command.
2. Java class meta-date inspection (SELECT).
3. Creation of PL/SQL function which call Java method. Execution of Java method.
4. Java class creation, loading and testing for SELECT and INSERT queries realization.
5. Conclusions (what seems good, what bad, what like, what is problematic).

# Java class creation and loading in database:

Text in blue and italic: SQL code

Objects in my Database

|  |
| --- |
| CREATE TYPE EMPLOYEE AS OBJECT(  employee\_id NUMBER,  first\_name VARCHAR2(30),  last\_name VARCHAR2(30),  salar NUMBER(8, 2)  ); |
| Result of execution: |

I used Eclipse IDE for creating java file but I think that it s not important which IDE we use. Reason why I used Eclipse is that a written code will have its own format like color of script or comment.

## Using LOADJAVA program

### 2.1.1 Define a class, Student, as follows:

|  |
| --- |
| **public** **class** Student {    **public** **static** String myStr() {  //Creating array from objects  String student[] = **new** String[5];      String myString = "";  //Filling array based on Object Construction  student[0] = "Doston";  student[1] = "Hamrakulov";  student[2] = "$1000000";  student[3] = "Samarkand";  student[4] = "Uzbekistan";  **for** (**int** i = 0; i < student.length; i++) {  myString = myString + student[i] + " ";  }  **return** myString;  }    **public** **static** **void** main(String[] args) {  Student st = **new** Student();  System.***out***.println(st.*myStr*());  }  } |

### 2.1.2 Compile the class on your client system using the standard Java compiler, as follows

|  |
| --- |
| >javac Student.java |
| Result of execution: |

### 2.1.3 Load the class on the server using loadjava. You must specify the user name and password. Run the loadjava command as follows:

|  |
| --- |
| loadjava -u HR/hr@localhost:1521/pdborcl -v -r -t Student.java |
| Result of execution: |

### 2.1.4 Publish the stored procedure through a call specification

In SQL Developer, connect to the database and define a top-level call specification for Student.myStr() as follows

|  |
| --- |
| CREATE OR REPLACE FUNCTION Student RETURN VARCHAR2 AS  LANGUAGE JAVA NAME 'Student.myStr () return java.lang.String';  / |
| Result of execution: |

### 2.1.5 Call the stored procedure, as follows:

|  |
| --- |
| VARIABLE myString VARCHAR2(50);  CALL Student() INTO :myString;  PRINT myString; |
| Result of execution: |

## 2.2 Using CREATE JAVA command.

### 2.2.1 Create Java a file named Billionaire in SQL Developer using CREATE JAVA

Use the CREATE JAVA statement to create a schema object containing a Java source, class, or resource.

|  |
| --- |
| CREATE or replace JAVA source named "Billionaire" AS  public class Billionaire {    public static String myStr() {  //Creating array from objects  String billionaire[] = new String[5];      String myString = "";  //Filling array based on Object Construction  billionaire[0] = "Doston";  billionaire[1] = "Hamrakulov";  billionaire[2] = "$1 000 000 000 000 000m";  billionaire[3] = "Samarkand";  billionaire[4] = "Uzbekistan";  for (int i = 0; i < billionaire.length; i++) {  myString = myString + billionaire[i] + " ";  }  return myString;  }    public static void main(String[] args) {  Billionaire st = new Billionaire();  System.out.println(st.myStr());  }  }  / |
| Result of execution: |

### 2.2.2 Publish the stored procedure through a call specification

In SQL Developer, connect to the database and define a top-level call specification for Billionaire.myStr() as follows

|  |
| --- |
| CREATE OR REPLACE FUNCTION Billionaire RETURN VARCHAR2 AS  LANGUAGE JAVA NAME 'Billionaire.myStr () return java.lang.String';  / |
| Result of execution: |

### 2.1.3 Call the stored procedure, as follows:

|  |
| --- |
| VARIABLE myString VARCHAR2(50);  CALL Student() INTO :myString;  PRINT myString; |
| Result of execution: |

## 2.3 Metadata

|  |
| --- |
| select OBJECT\_NAME, OBJECT\_TYPE, STATUS, CREATED,  GENERATED from USER\_OBJECTS  where CREATED >= TO\_DATE('25-11-2017', 'DD-MM-YYYY'); |
| Result of execution: |

# Java Class meta-data inspection (Data input) and class methods of execution SQL command (data output).

## About planning database schema

I am going to create 4 tables for this task which will use a simple system to store applications for scholarships and here I will make schema for the system.

Table:

* University
* Student
* Scholarship
* LineApplications

## 3.2 Creating database tables

After planning the database schema, create the database tables required by the schema plan. I will created 4 tables in sequence order:

|  |
| --- |
| CREATE TABLE University (  Univ\_id NUMBER(3) NOT NULL,  Title VARCHAR2(30) NOT NULL,  Street VARCHAR2(20) NOT NULL,  City VARCHAR2(20) NOT NULL,  State CHAR(2) NOT NULL,  Zip VARCHAR2(10) NOT NULL,  Phone VARCHAR2(12),  PRIMARY KEY (Univ\_id)  );  / |
| Result of execution: |

|  |
| --- |
| CREATE TABLE Applicant (  Applicant\_id NUMBER(3) NOT NULL,  Name VARCHAR2(30) NOT NULL,  Surname VARCHAR2(30) NOT NULL,  Country VARCHAR2(30) NOT NULL,  University NUMBER(3) REFERENCES University,  PRIMARY KEY (Applicant\_id)  );  / |
| Result of execution: |

|  |
| --- |
| CREATE TABLE Scholarship (  Scholar\_id NUMBER(4) PRIMARY KEY,  Description VARCHAR2(20),  Duration VARCHAR2(30),  Grant\_amount NUMBER(3)  );  / |
| Result of execution: |

|  |
| --- |
| CREATE TABLE LineApplications (  LineNo NUMBER(2),  Applicant\_id NUMBER(3) REFERENCES Applicant,  Scholar\_id NUMBER(4) REFERENCES Scholarship,  Deadline VARCHAR2(20),  PRIMARY KEY (LineNo, Applicant\_id)  );  / |
| Result of execution: |

## 3.3 Writing the Java classes

After creating the database tables, I consider the operations required in a scholarship applications database system and write the appropriate Java methods. In a simple system based on the tables defined in the preceding examples, I need methods for registering customers, stocking parts, entering orders, and so on. I can implement these methods in a Java class, ScholarshipManager, as follows:

|  |
| --- |
| **import** java.sql.\*;  **import** java.io.\*;  **import** oracle.jdbc.\*;  **public** **class** ScholarshipManager  {  **public** **static** **void** addUniversity (**int** univer\_id, String Title, String street,  String city, String state, String zipCode, String phoneNo) **throws** SQLException  {  String sql = "INSERT INTO Customers VALUES (?,?,?,?,?,?,?)";  **try**  {  Connection conn = DriverManager.*getConnection*("jdbc:default:connection:");  PreparedStatement pstmt = conn.prepareStatement(sql);  pstmt.setInt(1, univer\_id);  pstmt.setString(2, Title);  pstmt.setString(3, street);  pstmt.setString(4, city);  pstmt.setString(5, state);  pstmt.setString(6, zipCode);  pstmt.setString(7, phoneNo);  pstmt.executeUpdate();  pstmt.close();  }  **catch** (SQLException e)  {  System.***err***.println(e.getMessage());  }  }  **public** **static** **void** addScholarship (**int** scholar\_id, String description, String duration, **int** grant\_amount)  **throws** SQLException  {  String sql = "INSERT INTO StockItems VALUES (?,?,?,?)";  **try**  {  Connection conn = DriverManager.*getConnection*("jdbc:default:connection:");  PreparedStatement pstmt = conn.prepareStatement(sql);  pstmt.setInt(1, scholar\_id);  pstmt.setString(2, description);  pstmt.setString(3, duration);  pstmt.setInt(4, grant\_amount);  pstmt.executeUpdate();  pstmt.close();  }  **catch** (SQLException e)  {  System.***err***.println(e.getMessage());  }  }    **public** **static** **void** enterApplicant (**int** applicant\_id, String name, String surname, String country, **int** university) **throws** SQLException  {  String sql = "INSERT INTO Customers VALUES (?,?,?,?,?)";  **try**  {  Connection conn = DriverManager.*getConnection*("jdbc:default:connection:");  PreparedStatement pstmt = conn.prepareStatement(sql);  pstmt.setInt(1, applicant\_id);  pstmt.setString(2, name);  pstmt.setString(3, surname);  pstmt.setString(4, country);  pstmt.setInt(5, university);  pstmt.executeUpdate();  pstmt.close();  }  **catch** (SQLException e)  {  System.***err***.println(e.getMessage());  }  }  **public** **static** **void** addLineApplication (**int** lineNo, **int** applicant\_id, **int** scholar\_id, String deadline) **throws** SQLException  {  String sql = "INSERT INTO LineItems VALUES (?,?,?,?)";  **try**  {  Connection conn = DriverManager.*getConnection*("jdbc:default:connection:");  PreparedStatement pstmt = conn.prepareStatement(sql);  pstmt.setInt(1, lineNo);  pstmt.setInt(2, applicant\_id);  pstmt.setInt(3, scholar\_id);  pstmt.setString(4, deadline);  pstmt.executeUpdate();  pstmt.close();  }  **catch** (SQLException e)  {  System.***err***.println(e.getMessage());  }  }  **public** **static** **void** totalApplications () **throws** SQLException  {    String sql = "SELECT A.Applicant\_id, ROUND(SUM(S.Grant\_amount \* S.Grant\_amount)) AS TOTAL " +  "FROM Applicant A, LineApplications L, Scholarship S " +  "WHERE A.Applicant\_id = L.Applicant\_id AND L.Scholar\_id = S.Scholar\_id " +  "GROUP BY A.Applicant\_id";  **try**  {  Connection conn = DriverManager.*getConnection*("jdbc:default:connection:");  PreparedStatement pstmt = conn.prepareStatement(sql);  ResultSet rset = pstmt.executeQuery();  **int** count = 0;  **while** (rset.next()) {  ++count;    }  **if** (count == 0) {  System.***out***.println("No data found");  }  System.***out***.println(count);  rset.close();  pstmt.close();  }  **catch** (SQLException e)  {  System.***err***.println(e.getMessage());  }  }  **public** **static** **void** deleteApplication (**int** applicantNo) **throws** SQLException  {  String sql = "DELETE FROM LineApplications WHERE Applicant\_id = ?";  **try**  {  Connection conn = DriverManager.*getConnection*("jdbc:default:connection:");  PreparedStatement pstmt = conn.prepareStatement(sql);  pstmt.setInt(1, applicantNo);  pstmt.executeUpdate();  sql = "DELETE FROM Applicant WHERE Applicant\_id = ?";  pstmt = conn.prepareStatement(sql);  pstmt.setInt(1, applicantNo);  pstmt.executeUpdate();  pstmt.close();  }  **catch** (SQLException e)  {  System.***err***.println(e.getMessage());  }  }  } |
| Result of execution: |

## Loading the Java classes

After writing the Java classes, use the loadjava tool to upload my Java stored procedures into Oracle Database, as follows:

|  |
| --- |
| loadjava -u HR/hr@localhost:1521/pdborcl -v -r -t Scholarshipmanager.java |
| Result of execution: |

## 3.5 Publishing the Java classes

After loading the Java classes, publish my Java stored procedures in the Oracle data dictionary. To do this, I must write call specifications that map Java method names, parameter types, and return types to their SQL counterparts.

The methods in the ScholarshipManager Java class are logically related. I can group their call specifications in a PL/SQL package. To do this, first, let`s create the package specification, as follows:

|  |
| --- |
| CREATE OR REPLACE PACKAGE scholar\_mgr AUTHID CURRENT\_USER AS  PROCEDURE add\_university (univ\_id NUMBER, title VARCHAR2,  street VARCHAR2, city VARCHAR2, state CHAR, zip VARCHAR2, phone VARCHAR2);  PROCEDURE add\_scholarship (scholar\_id NUMBER, description VARCHAR2,  duration VARCHAR2, grant\_amount NUMBER);  PROCEDURE enter\_applicant (applicant\_id NUMBER, name VARCHAR2,  surname VARCHAR2, country VARCHAR2, university NUMBER);  PROCEDURE add\_line\_application( lineNo NUMBER, applicant\_id NUMBER,  scholar\_id NUMBER, deadline VARCHAR);  PROCEDURE total\_applications;  PROCEDURE delete\_application (applicant\_id NUMBER);  END scholar\_mgr;  / |
| Result of execution: |

Then, let`s create the package body by writing call specifications for the Java methods, as follows:

|  |
| --- |
| CREATE OR REPLACE PACKAGE BODY scholar\_mgr AS  --University  PROCEDURE add\_university (univ\_id NUMBER, title VARCHAR2,  street VARCHAR2, city VARCHAR2, state CHAR, zip VARCHAR2, phone VARCHAR2) AS LANGUAGE JAVA  NAME 'ScholarshipManager.addUniversity(int, java.lang.String, java.lang.String,  java.lang.String, java.lang.String, java.lang.String, java.lang.String)';  --Scholarship  PROCEDURE add\_scholarship (scholar\_id NUMBER, description VARCHAR2,  duration VARCHAR2, grant\_amount NUMBER) AS LANGUAGE JAVA  NAME 'ScholarshipManager.addScholarship(int, java.lang.String, java.lang.String, int)';  --Applicant  PROCEDURE enter\_applicant (applicant\_id NUMBER, name VARCHAR2,  surname VARCHAR2, country VARCHAR2, university NUMBER) AS LANGUAGE JAVA  NAME 'ScholarshipManager.enterApplicant(int, java.lang.String, java.lang.String, java.lang.String, int)';  --LineApplications  PROCEDURE add\_line\_application(lineNo NUMBER, applicant\_id NUMBER,  scholar\_id NUMBER, deadline VARCHAR) AS LANGUAGE JAVA  NAME 'ScholarshipManager.addLineApplication(int, int, int, java.lang.String)';  --total\_application for totalApplication method in Java  PROCEDURE total\_applications AS LANGUAGE JAVA  NAME 'ScholarshipManager.totalApplications()';  --for delete application method in Java  PROCEDURE delete\_application (applicant\_id NUMBER) AS LANGUAGE JAVA  NAME 'ScholarshipManager.deleteApplication(int)';  END scholar\_mgr; |
| Result of execution: |

## 3.6 Calling the Java stored Procedure

After publishing the Java classes, call my Java stored procedures from the top level and from database triggers, SQL data manipulation language (DML) statements, and PL/SQL blocks.

From an anonymous PL/SQL block, I may start applying for new scholarship in the system by implementing parts, as follows:

|  |
| --- |
| BEGIN  scholar\_mgr.add\_scholarship(2010, 'New UzbekGrant', '6 month', 900);  scholar\_mgr.add\_scholarship(2011, 'Young UzbekGrant', '6 month', 800);  scholar\_mgr.add\_scholarship(2012, 'New UzbekGrant', '6 month', 900);  scholar\_mgr.add\_scholarship(2013, 'Korean Grant', '6 month', 900);  scholar\_mgr.add\_scholarship(2014, 'German Grant', '6 month', 100);  scholar\_mgr.add\_scholarship(2015, 'Samarkand Grant', '6 month', 300);  scholar\_mgr.add\_scholarship(2016, 'Latvian Grant', '2 month', 500);  COMMIT;  END; |
| Result of execution: |
| BEGIN  scholar\_mgr.add\_university(100, 'Uzbek National Uni', 'Beruniy 12', 'Tashkent', 'UZ', '07000', '+99893777703');  scholar\_mgr.add\_university(101, 'Westmenester', 'Nurota 34', 'Samarkand', 'UZ', '04450', '+99893777703');  scholar\_mgr.add\_university(102, 'RTU', 'KALKU 1', 'RIGA', 'LV', '07000', '+371232323');  COMMIT;  END; |
| Result of execution: |

|  |
| --- |
| BEGIN  scholar\_mgr.enter\_applicant(500, 'Doston', 'Hamrakulov', 'Uzbekistan', 100);  scholar\_mgr.add\_line\_application(01, 500, 2010, '21-SEP-2019');  scholar\_mgr.add\_line\_application(02, 500, 2012, '21-SEP-2019');  scholar\_mgr.add\_line\_application(03, 500, 2015, '21-SEP-2019');    scholar\_mgr.enter\_applicant(501, 'Orif', 'Doniyarov', 'Uzbekistan', 101);  scholar\_mgr.add\_line\_application(04, 501, 2012, '21-SEP-2019');  scholar\_mgr.add\_line\_application(05, 501, 2013, '21-DEC-2019');  scholar\_mgr.add\_line\_application(06, 501, 2016, '21-OCT-2019');    scholar\_mgr.enter\_applicant(502, 'John', 'Smith', 'UK', 102);  scholar\_mgr.add\_line\_application(07, 502, 2012, '21-JUN-2018');  scholar\_mgr.add\_line\_application(08, 502, 2011, '21-JUN-2018');  scholar\_mgr.add\_line\_application(09, 502, 2013, '21-JUN-2018');  COMMIT;  END; |
| Result of execution: |

Final result:

|  |
| --- |
| SET SERVEROUTPUT ON  CALL dbms\_java.set\_output(2000);  CALL scholar\_mgr.total\_applications(); |
| Result of execution: |

# Defining methods with the Java class and its application

**In Paragraph 1.1**, when I demonstrated **LOADJAVA program**, I used function creation with all details.

# Conclusions.

To be honest, this task was more difficult for me rather than previous one. Because I faced many problems, such as

* ORA-00942: table or view does not exist pl\*sql stored procedure
* ORA-29541: class could not be resolved
* others

Finally, I solved all error and got the result after spending at LONG TIME on these issues.

To sum up, I have gained a lot of skill of Oracle, Pl/SQL and usage of JDBC driver how to connect it with PL/SQL and execute statements. But there are more things which I should learn. Therefore, I think I will improve my knowledge till the ending of semester by doing next tasks and in lectures.