Semester project TDS II.

Below you will find an outline for developing a semester project, noting that you must have pieces of code for all required tasks (if your structure does not support this, this needs to be supplemented). It is necessary to incorporate all the commands and options that are included in the OA PL/SQL course into the project. The list below is only a rough outline of what should appear in the project (section numbers may not correspond to the new course in the online environment). Students who are enrolled in TDS I should first hand in the project from TDS I and then expand it with the requirements from TDS II.

Use Oracle SQL Data Modeler and SQL Developer to develop, with the understanding that everything must work on the school's Oracle server, for which I will send you the login information.

- Lesson 1
  - Anonymous procedure – containing:
  - about the Declaration
  - Block
  - Exception
- Lesson 2
- Add your own variables in the declaration:
  - Variables of type String, Integer, Date, BOOLEAN
  - Define a constant
  - Define a variable with a default value
  - By transformation using %TYPE
  - Using implicit explicit data type conversion
  - Output of results to the console (DBMS.PUT_LINE)
  - Write a nested procedure (at least one nesting)
- Lesson 3
  - Write a procedure that will contain INSERT, UPDATE, DELETE, MERGE
  - Use in procedure SELECT atr INTO prom …
  - Use SQL%FOUND, SQL%NOTFOUND, SQL%ROWCOUNT as part of data manipulation
  - Use COMMIT, ROLLBACK, SAVEPOINT within the transaction handling procedure
- Lesson 4
  - Implement a procedure with IF-THEN-ELSE and IF-ELSEIF-ELSE
  - Implement a procedure with CASE-WHEN-ELSE (with a variable after the CASE) as a separate implementation, or as the result of an assignment to a variable.
  - Implement CASE-WHEN-ELSE with a condition after WHEN
  - Implement a procedure solving all variants of combinations of logical operators AND, OR, NOT with input values TRUE, FALSE, NULL
  - Implement a loop with LOOP – EXIT [WHEN] – END LOOP
  - Implement a WHILE loop - LOOP condition - END LOOP
  - Implement the cycle FOR counter IN lower..upper LOOP – END LOOP
  - Implement a loop with a reverse counter REVERSE
  - Implement nested loops with EXIT termination and labels
- Lesson 5
  - Implement an explicit cursor in the CURSOR declaration in the body of the OPEN-FETCH-CLOSE procedure

    - Implement a cursor with loading into an INTO record in the %ROWTYPE declaration Use the %ISOPEN, %NOTFOUND, %FOUND, %ROWCOUNT flags in the cursor implementation
    - Implement a cursor FOR record name LOOP cursor name END LOOP also use EXIT WHEN inside the cursor
    - Implement a cursor with parameters to be entered in the body of the procedure CURSOR name (parameters) IS …
    - Implement cursor for UPDATE with WAIT and NOWAIT
    - Implement multiple nested cursors, the inner cursor is affected by the outer cursor parameter
- Lesson 6
    - Implement user defined %ROWTYPE records taken from table
    - Definition of custom record type TYPE data type IS RECORD (…)
    - Table (field) INDEXED BY TABLE and INDEXED BY TABLE OF RECORDS, INDEX BY BINARY_INTEGER
- Lesson 7
    - EXCEPTION WHEN … THEN exception handling with WHEN OTHERS THEN
    - Catching Oracle server exceptions predefined (constants), undefined (numbered errors)
    - Declaration of your own exception name DECLARATION exception name EXCEPTION, PRAGMA EXCEPTION INIT …
    - Error status listing and exception comments: SQLCODE, SQLERRM
    - User-defined exceptions RAISE_APPLICATION_ERROR with a defined name and the second variant with the number of your own exception and a comment directly in the body of the procedure
    - Calling an exception in a nested procedure and handling it in the nested or in the parent
- Lesson 8
    - Creating a procedure - you can also create previous commands as nested procedures
    - Use of parameters for IN, OUT, IN OUT procedures
    - Try what happens when you put some value in the OUT parameter and it wants to work with it in the body of the procedure
    - Assignment of parameters by order or assignment by name parameter name => parameter value, using the DEFAULT value for the procedure parameter
- Lesson 9
    - Create a function with a return value CREATE OR REPLACE FUNCTION function name (param,…) RETURN data type IS …
    - Use your defined function directly in the SQL command, in the SELECT, WHERE, GROUP BY, ORDER BY section
    - View the data dictionary with your sources, USER_TABLES, USER_IDEXES, USER_SOURCES, USER OBJECTS
    - Write two procedures that call each other and throw exceptions
    - Verify authorization to manipulate individual objects TABLES, SEQUENCES, VIEWS, PROCEDURES
    - Try assigning user rights for the above objects and allowed operations
- Lesson 10
    - Create a PACKAGE, with previously implemented procedures and functions PACKAGE (header), PACKAGE BODY (body)
    - Overloading of subroutines - define a PACKAGE where one procedure (function) will have several interpretations (executions), according to the number and type of input parameters Implements a FUNCTION that will have any number of text parameters that will then be combined in reverse order together, or come up with another alternative function (procedure)
- Lesson 11
        - Try the prome's persistent state

- (global variables) in PACKAGE
    - Use a PACKAGE supported (offered) by ORACLE, for each listed (not often used) PACKAGE (DBMS_LOB, DBMS_LOCK, DBMS_OUTPUT, http, UTL_FILE, UTL_MAIL, DBMS_SCHEDULER) try its functions

• Lesson 12
    - Use dynamically generated SQL using PACKAGE DBMS_SQL Implement EXECUTE IMMEDIATE with a separate code variant, INTO parameter passing and USING IN OUR param definition of input/output parameters, …
    - Try ALTER PROCEDURE … COMPILE and the same for FUNCTION, PACKAGE SPECIFICATION AND PACKAGE BODY
    - Implement procedures to increase the performance of your procedures, show an example of speeding up execution with and without using: NOCOPY, DETERMINISTIC, FORALL, BULK COLLECT, SELECT, FETCH and RETURNING

• Lesson 13
    - Create several triggers in the variants BEFORE AFTER FOR EACH ROW
    - Create a trigger on a composite view (composed of multiple tables), insert a new record into the view and use INSTEAD OF to insert data into the source tables
    - Write a trigger using :NEW and :OLD variables
    - Write a single trigger with branching for multiple database operations, for example: BEFORE INSERT OR UPDATE OR DELETE ON tab … IF DELETEING THEN …
    - Write a trigger that will generate the problem of mutating tables (Mutation-error)

• Lesson 14
    - Explore dependencies between objects, such as procedures calling each other, views from tables, etc.
    - List the dependency machine, for example from a table, a view is created that I use in one procedure and it is subsequently used in another procedure
    - Describe dependency options with TIMESTAMP and SIGNATURE (REMOTE_DEPENDENCIES_MODE = SIGNATURE)

• Lesson 15
    - Test the Oracle optimization parameters, PLSQL_CODE_TYPE, PLSQL_OPTIMIZATIUON_LEVEL, and list their settings from the USER_PLSQL_OBJECT_SETTING view
    - Try different levels of optimization depending on the execution speed of a complex procedure
    - Try listing compiler warnings PSQL_WARNINGS, DBMS_WARNINGS
    - Try conditional compilation depending on Oracle SQL Server version, DBMS_DB_VERSION, $IF $END
    - Try to hide the DBMS_DDL.CREATE_WRAPPED source code for one procedure of your choice