Students' Guide

Requirements of your homework

During the SQL labs you should create SQL scripts, which correspond to the SQL script skeleton provided. **In the case of the SQL1 lab, you should also hand in a PDF report.** In the script skeleton it can easily be identified, which solution belongs to which task. The task numbers found in the script should logically be substituted with the ones found on the exercise sheet. (The exercise numbers of the SQL1 lab contain a dot, while those of the SQL2 and SQL3 lab do not. Do not be confused.) **If the homework you hand in does not correspond to the skeleton, you solution will not be evaluated, so pay attention!** If the exercise is ambigous for you, contact your lab instructor **before** handing in your homework.

For the SQL2 and SQL3 labs, the initialization scripts which create and fill the tables, can be downloaded from our website. Exercise number zero in this case is to download and execute the init script. The best practice to save the init script is to open it in the browser, copy, and paste its contents to SQL Developer, and save it. You should run the script by pressing F5 in SQL Developer.

You homework should contain two files for the SQL1 lab:

- `<neptun>-2-<groupcode>.sql`, your SQL script, solving the tasks
- `<neptun>-2-<groupcode>.pdf`, your report mainly with text content. This should contain the high-quality documentation of your work. The report should unambiguously and explicitly contain the solutions you gave for the different tasks. It should contain the ER model you designed and the relational schema you turned it into.

For the SQL2 lab, you should hand in two files:

- `<neptun>-4-<groupcode>.sql`, your script containing the solutions.
- `<neptun>-4-<groupcode>.pdf`, your report mainly with text content. This should contain the high-quality documentation of your work. The report should unambiguously and explicitly contain the solutions you gave for the different tasks and explain them.

For the SQL2 lab, you should hand in two files:

- `<neptun>-5-<groupcode>.sql`, your script containing the solutions.
- `<neptun>-5-<groupcode>.pdf`, your report mainly with text content. This should contain the high-quality documentation of your work. The report should unambiguously and explicitly contain the solutions you gave for the different tasks and explain them.

In the above file names, you should replace `<neptun>` with your actual NEPTUN code i.e. ABC123, and `<groupcode>` with the same group code as in case of the Oracle lab.

Functional and formal requirements of your SQL script:

Az SQL szkript formai és funkcionális megkötései:

- Your script should be encoded with UTF-8.
- Your script should correspond to the provided skeleton and contain the solutions in the same order as on the exercise sheet.
- The script should NOT call the init script (for the SQL1 lab there is no init script, we first will use the init script in the SQL2 lab).
- For SQL2 and SQL3 labs: If the init script and your script is executed multiple times in this order, we should get the same results. To achieve this, at the beginning of your script, it should drop all your own objects (DROP TABLE, etc....). The init script takes care of dropping the official objects but your objects should be dropped by you.
- For the SQL1 lab: If the solution script is executed multiple times in a row, we should get the same result each time, so drop your own objects (DROP TABLE, etc....) at the beginning of your script.
- The script must not use procedural PL/SQL elements. Whatever can be found in the Oracle 12c SQL reference (http://docs.oracle.com/database/121/SQLRF/toc.htm,), is usually not a procedural element. This reference however, references the PL/SQL reference, which contains procedural elements as well. (For example, CREATE PROCEDURE is included in the SQL reference, but refers the PL/SQL reference right away)
- The SQL statements of the script MUST NOT use schema names before object names. Result columns should only be renamed if the task explicitly asks you so or implies it (by – for example – providing the name of the result column).
- The script should produce a solution in XML format. To achieve this, you should modify the skeleton in the following way:
    - Before `prompt <tasks>` drop your own objects.
    - The solution of tasks should come between `prompt <tasks>` and `prompt </tasks>`, in the same order as on the exercise sheer.
    - The solution of each task should be preceded by the two lines below. Attribute n is the number of the task from the exercise sheet. In this example it is 1.1 but it should be updated for each task to match **the exact same** task number as on the exercise sheet. (In case of labs SQL2 and SQL3, the format of the task number if x.y that is two numbers and a dot between them, while in the case of lab SQL1 it is a single number so in the latter case you should use this single number as the value of attribute n)

      ```
      prompt <task n="1.1">
      prompt <![CDATA[
      ```
    - Then should come the SQL statement

- After the SQL statement, come the following two lines (same for each task):
  `prompt ]]>`
  `prompt </task>`
- The opening and closing `prompt` messages of unsolved tasks should not appear in the script, you should remove these instead.
- The script should execute statement `set feedback on` right before the data manipulation exercises and statement `set feedback off` right after them.

- Before handing in your final script, make sure that it properly works in SQLcl. Before running your own script in SQLcl, in case of the SQL2 and SQL3 labs, execute the init script **manually**. In the case of the SQL1 lab, you should not execute the init script before your own script. SQLcl can be downloaded from the course website and Oracle's website. SQLcl can be started the following way:

  `sql /nolog`

  After this, you can connect to the DB server the following way:

  `CONNECT username@//rapid.eik.bme.hu:1521/szglab`

  It might be useful to know that the result of your queries can be saved to a text file using the `SPOOL <filename>` command in SQLcl.
- Using the above method in SQLcl, check whether your script really returns a well-formed XML output with the `<tasks>` tag as its root element, which contains `<task>` elements. Within the `<task>` element, the output of the given task must appear the following way: `<![CDATA[output]]>`.
- The SQL script itself must be well-formed between its `<tasks>` and `prompt </tasks>` elements.
- For checking whether an XML file is well-formed, do the following:
  - Cut the part of the script between `<tasks>` amd `prompt </tasks>`, put it in an XML file, and open this XML file in a browser (Firefox, Chrome or IE). If the result appears in a hierarchical **tree structure** then it is well-formed. If not, ot we get an error then it is not well-formed.
  - Open the result of our script similarly, and check whether it is well-formed!
- **If the script does not fulfill the above requirements, the homework cannot be evaluated!**

Furthermore, make sure that your solution avoids conceptional mistakes or mistakes affecting the efficiency, for example:

- using an embedded SELECT instead of HAVING
- using an embedded SELECT instead of a join
- using unnecessary set operators
- not using outer join when needed

- comparing the NULL value without using IS NULL or IS NOT NULL

Finally, a note: In many cases, the exercise asks for solving the given problem without listing the column names. This does not mean that column names should not appear on the output. What it means is that in the query, these column names shouldn't be listed.

A few SQL tips

- Each query contains the necessary keywords in the following order: SELECT – FROM – WHERE – GROUP BY – HAVING – ORDER BY (not all of these are mandatory of course), except(!) for hierarchical queries.
- In relational algebra, SELECT – FROM – WHERE (mostly) correspond to projection – cartesian product – selection, respectively. The difference is that while relational algebra operators are set operators (that is, the result does not contain repetitions), in the case of SQL, SELECT DISTINCT has to be used instead of SELECT to filter the repetitions.
- There is a huge difference between HAVING and WHERE. HAVING always(!) defines a condition related to an aggregation (SUM, AVG, COUNT, etc.).
- PRIMARY KEY = UNIQUE + NOT NULL, but while there is only one PRIMARY KEY in each schema, there can be any number of UNIQUE attributes.
- If an attribute is UNIQUE, it still can be NULL.
- The values of NULL = NULL, and NULL <> NULL are both false (more precisely UNKNOWN) and thus, will not be included in the result set.
- UNION by default, returns a set of DISTINCT elements, and this way is the only SQL statement returning a real set by default. If we would like to see repetitions, we should use UNION ALL instead.
- Every single non-aggregated attribute listed after SELECT has to be listed after GROUP BY as well. Otherwise, the query will not be compiled.
- There is a big difference between CHAR, and VARCHAR2. The formerhas a fixed size while the latter is a string with a dynamic length. The difference is significant during pattern matching since in case of CHAR(5), three-letter words ending with L actually end with two SPACE characters.
- Functions TO_DATE and TO_CHAR are described in detail in Oracle's SQL handbook: http://docs.oracle.com/database/121/SQLRF/functions.htm#SQLRF006.
- Keyword AS is not mandatory in the SELECT list. Moreover, when renaming tables in the FROM list it must not be used.
- Embedded queries (in parentheses) can be used between the FROM and WHERE keywords.

- When using quotation marks, the tables and attribute names stored in the database are mostly (but not always) evaluated in a case-sensitive manner. It is better to avoid using them when creating and inserting into tables. Also, they can always be omitted.
- Table names should not be preceded by schema names (e.g. the user name). Also, when unambiguous, table names do not have to be used before attribute names.
- The „(+)" sign of the outer join operation should always be put next to the attribute which can include NULL values.
- In case of an outer join, „(+)" has to be included after each of those attributes which belongs to the table supplemented with NULL values. This is not true for multiple outer joins. See the following example for explanation:

```
select a.id, b.id, c.id
  from a, b, c
 where
    -- join
    a.id = b.a_id (+)
    and b.c_id = c.id (+)
    -- filtering, here (+) is needed at every single
    -- occurrence of b and c
    and b.color (+) in ('red', 'green')
;
```

- In a hierarchical query, keyword PRIOR always precedes that attribute which belongs to the element that other elements refer with the value of the given attribute. (These referring elements are listed in the next iteration, so PRIOR points back to the element listed first.) PRIOR binds to an attribute se it cannot be used in the scope of an operator or function, it can only be followed by an attribute name.
- Function NVL2 might be useful. It can be parameterized as follows: NVL2(attribute name, output in case of a non-NULL value, output in case of a NULL value)
- NVL and DECODE will surely have to be used in some tasks, so it's worth looking these up as well.