School of Computing and Information Technology
University of Wollongong

Session: **Summer 2020**
Lecturer: **Janusz R. Getta**

CSCI835 Database Systems
Assignment 2
27 June 2020

Scope

This assignment includes the tasks related to indexing of relational tables, verification of consistency constraints with a stored procedure, and automatic modification of derived data with a statement trigger.

The outcomes of the assignment are due by **Saturday, 4 July, 2020, 9.00 pm (sharp).**

Please read very carefully information listed below.

This laboratory contributes to 13% of the total evaluation in a subject CSCI835 Database Systems.

A submission procedure is explained at the end of specification.

This assignment consists of 3 tasks and specification of each task starts from a new page.

It is recommended to solve the problems before attending the laboratory classes in order to efficiently use supervised laboratory time.

A submission marked by Moodle as "late" is treated as a late submission no matter how many seconds it is late.

A policy regarding late submissions is included in the subject outline.

A submission of compressed files (zipped, gzipped, rared, tared, 7-zipped, lhzed, … etc) is not allowed. The compressed files will not be evaluated.

All files left on Moodle in a state "`Draft(not submitted)`" will not be evaluated.

An implementation that does not compile due to one or more syntactical errors scores no marks.

It is expected that all tasks included within **Assignment 2** will be solved **individually without any cooperation** with the other students. If you have any doubts, questions, etc. please consult your lecturer or tutor during the laboratory classes. Plagiarism will result in a **FAIL** grade being recorded for the assessment task.

Prologue

Download the files `dbcreate.sql`, `dbload.sql`, and `dbdrop.sql` included in a section SAMPLE DATABASES on Moodle. A conceptual schema of a sample database is available in the same section on Moodle. It is recommended to familiarize yourself with a conceptual schema and implementations of relational tables.

SQL script `dbcreate.sql` creates a sample database. To drop a sample database, you can use a script `dbdrop.sql`. To load data into a sample database, process as script `dbcreate.sql`. It is strongly recommended to drop a sample database and to re-create it before implementation each task.

Connect to Oracle database server and process the following SQL statement that saves a query processing plan for a given `SELECT` statement in `PLAN_TABLE`.

```
EXPLAIN PLAN FOR SELECT ORDER_ID, ORDER_DATE FROM ORDERS;
```

Next, process the following `SELECT` statement to display a query processing plan stored in `PLAN_TABLE`.

```
SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);
```

Among the others, you should get the following results.

```
PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
Plan hash value: 1275100350

-------------------------------------------------------------------------------
| Id  | Operation         | Name   | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |        |   131 |  2882 |     3   (0)| 00:00:01 |
|   1 |  TABLE ACCESS FULL| ORDERS |   131 |  2882 |     3   (0)| 00:00:01 |
-------------------------------------------------------------------------------
```

A line `TABLE ACCESS FULL| ORDERS` in a plan given above indicates that a database system plans to access a table `ORDERS` to compute the query.

Next, create an index on the columns `ORDER_ID` and `ORDER_DATE` in a relational table `ORDERS`.

```
CREATE INDEX ORDERS_IDX ON ORDERS(ORDER_ID, ORDER_DATE);
```

Again, process the following SQL statements that save a query processing plan for the same `SELECT` statement as before in `PLAN_TABLE` and display a query processing plan stored in `PLAN_TABLE`.

```
EXPLAIN PLAN FOR SELECT ORDER_ID, ORDER_DATE FROM ORDERS;
```

Among the others, you should get the following results.

```
PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------
Plan hash value: 2467194144

-------------------------------------------------------------------------------
| Id  | Operation        | Name       | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT |            |   131 |  2882 |     1   (0)| 00:00:01 |
|   1 |  INDEX FULL SCAN | ORDERS_IDX |   131 |  2882 |     1   (0)| 00:00:01 |
-------------------------------------------------------------------------------
```

This time a database system plans to use an index `ORDERS_IDX` created a moment ago to process the same query. Note, a line `INDEX FULL SCAN  | ORDERS_IDX` in a plan given above means that a database system plans to horizontally traverse leaf level an index `ORDERS_IDX` to find the values in the columns `ORDER_ID` and `ORDER_DATE`.

Conclusions

`EXPLAIN PLAN` statement of SQL can be used to get information about a processing plan created by a query processor for a given `SELECT` statement. A query processing plan provides information on whether and index created earlier will be used for processing of SQL statement. We shall use `EXPLAIN PLAN` statement to check whether an index created to speed up `SELECT` statement will be used for processing of the statement.

To drop an index, process a statement

```
DROP INDEX ORDERS_IDX;
```

No report is expected from the actions performed in the Prologue.

Tasks

Task 1 (3 marks)

Indexing relational tables

Your task is to find what indexes should be created to speed up processing of `SELECT` statements listed below. You are expected to create one index for one `SELECT` statement. To simplify the problem, assume that any index which is later on used by a query processor to speed up processing of `SELECT` statement will do.

(i)
```
SELECT *
FROM ORDER_DETAIL
WHERE PRODUCT_NAME = 'BOLT' AND
      QUANTITY > 100;
```

(ii)
```
SELECT DISTINCT CATEGORY_NAME
FROM PRODUCT;
```

(iii)
```
SELECT UNIT_PRICE
FROM ORDER_DETAIL
WHERE QUANTITY IN (100, 200, 300) OR
      DISCOUNT = 0.01;
```

(iv)
```
SELECT CATEGORY_NAME, SUPPLIER_NAME, COUNT(*)
FROM PRODUCT
GROUP BY CATEGORY_NAME, SUPPLIER_NAME;
```

(v)
```
SELECT SUPPLIER_NAME, UNIT_PRICE
FROM PRODUCT
ORDER BY UNIT_PRICE, QUANTITY_PER_UNIT;
```

(vi)
```
SELECT SUPPLIER.COMPANY_NAME, SUPPLIER.CITY
FROM PRODUCT JOIN SUPPLIER
     ON PRODUCT.SUPPLIER_NAME = SUPPLIER.COMPANY_NAME;
```

Implement SQL script `solution1.sql` such that for each one of `SELECT` statements given above the script performs the following actions.

(i) Find and list a query processing plan for `SELECT` statement without an index.
(ii) Create an index.
(iii) Find and list a query processing plan for `SELECT` statement with an index.
(iv) Drop an index.

When ready process SQL script file `solution1.sql` and save a report from processing in a file `solution1.lst`.

Your report must include a listing of all PL/SQL statements processed. To achieve that put the following SQL*Plus commands:

```
SPOOL <a-path-to-location-of>\solution1
SET ECHO ON
SET FEEDBACK ON
SET LINESIZE 100
SET PAGESIZE 200
SET SERVEROUTPUT ON
```

at the beginning of SQL script and

```
SPOOL OFF
```

at the end of SQL script. Note, that a symbol `<a-path-to-location-of>` represent a path to location of a file `solution1.lst` with a report. For example, a command

```
SPOOL C:\CSCI835\assignment2\solution1
```

saves a report `solution1.lst` from processing of a script `solution1.sql` at location `C:\CSCI835\assignment2` .

Process your script with SQL Developer. A report from processing of a script can be found in a file `<a-path-to-location-of>\solution1.lst` . It is also possible to save a report in a file `solution1.lst` using a Save File button in Script Output tab of SQL Developer.

Deliverables
A file `solution1.lst` with a report from processing of SQL script `solution1.sql`. A report must have no errors and it must list all SQL statements processed.

Task 2 (5 marks)
Enforcing a consistency constraint on data entry
Implement a stored PL/SQL procedure

```
INSERT_ORDER_DETAIL(order_id,product_name,unit_price,quantity,discount)
```

that inserts a row into a relational table `ORDER_DETAIL` and enforces the following consistency constraint on data entry into a relational table `ORDER_DETAIL`.

A product can be ordered only if it is not discontinued.

If the consistency constraint is satisfied insert and commit a row in `ORDER_DETAIL` table. Otherwise, use `DBMS_OUTPUT` PL/SQL package to display an error message when the consistency constraint is violated and do not insert a row.

When `INSERT_ORDER_DETAIL` procedure is ready create SQL script `solution2.sql` that stores the procedure in a data dictionary and tests the procedure with two `EXECUTE` statements. First, test the procedure for a product that is not discontinued and then test it again for a product that is discontinued. Any discontinued and not discontinued products used for testing will do.

Process SQL script `solution2.sql` and save a report from processing in a file `solution2.lst`.

Your report must include a listing of all PL/SQL statements processed. To achieve that put the following SQL*Plus commands:

```
SPOOL <a-path-to-location-of>\solution2
SET ECHO ON
SET FEEDBACK ON
SET LINESIZE 100
SET PAGESIZE 200
SET SERVEROUTPUT ON
```

at the beginning of SQL script and

```
SPOOL OFF
```

at the end of SQL script. Note, that a symbol `<a-path-to-location-of>` represent a path to location of a file `solution2.lst` with a report. For example, a command

```
SPOOL C:\CSCI835\assignment2\solution2
```

saves a report `solution2.lst` from processing of a script `solution2.sql` at location `C:\CSCI835\assignment2` .

Process your script with SQL Developer. A report from processing of a script can be found in a file `<a-path-to-location-of>\solution2.lst` . It is also possible to save a report in a file `solution2.lst` using a Save File button in Script Output tab of SQL Developer.

Deliverables
A file `solution2.lst` with a report from processing of SQL script `solution2.sql`. A report must have no errors and it must list all PL/SQL and SQL statements processed.

Task 3 (5 marks)
Automatic update of a derived attribute

It is recommended to process the scripts `dbdrop.sql`, `dbcreate.sql`, and `dbload.sql` to refresh a sample database after testing of the previous task.

Modify a structure of the sample database such that after the modifications it is possible to store for each product information about the total number of times a product has been ordered by the customers. It is important to find the best design. Note, that a product could be registered a moment ago and it has not been ordered by any customers yet. Hence, remember to enforce appropriate consistency constraints.

Next, set up the values in the new column such that information about the total number of times each product has been ordered by the customers is consistent with the present contents of the database.

Next, implement **a statement database trigger** that automatically changes the total number of times a product has been ordered whenever a product is added to an order or a product is removed from an order or a product is replaced with another product in an order.

Create SQL script `solution3.sql` with SQL statements that:

(i) perform a structural modification of a sample database such that for each product it is possible to store information about the total number of times a product has been ordered so far.

(ii) modify the contents of a sample database such that information about the total number of times each product has been ordered so far is included in the database and such information is consistent with the present contents of the database.

(iii) create a statement trigger that automatically changes the total number of times a product has been ordered whenever a product is added to an order or a product is removed from an order or a product is replaced with another product in an order.

(iv) comprehensively test the trigger and display the modifications performed by the trigger after each test. Display only the modifications performed by the trigger.

When ready process SQL script `solution3.sql` and record the results of processing in a file `solution3.lst`. Your report must include a listing of all PL/SQL statements processed.

To achieve that put the following SQL*Plus commands:

```
SPOOL <a-path-to-location-of>\solution3
SET ECHO ON
SET FEEDBACK ON
```

```
SET LINESIZE 100
SET PAGESIZE 200
SET SERVEROUTPUT ON
```

at the beginning of SQL script and

```
SPOOL OFF
```

at the end of SQL script. Note, that a symbol `<a-path-to-location-of>` represent a path to location of a file `solution3.lst` with a report. For example, a command

```
SPOOL C:\CSCI835\assignment2\solution3
```

saves a report `solution2.lst` from processing of a script `solution3.sql` at location `C:\CSCI835\assignment3`.

Process your script with SQL Developer. A report from processing of a script can be found in a file `<a-path-to-location-of>\solution3.lst` . It is also possible to save a report in a file `solution2.lst` using a Save File button in Script Output tab of SQL Developer.

Deliverables
A file `solution3.lst` with a report from creating and testing of a database trigger that automatically changes information about the total number of times a product has been ordered by the customers. A report must have no errors and it must list all SQL statements processed.

Submission

Submit the files **solution1.lst**, **solution2.lst**, and **solution3.lst** through Moodle in the following way:

(1) Access Moodle at **http://moodle.uowplatform.edu.au/**
(2) To login use a **Login** link located in the right upper corner the Web page or in the middle of the bottom of the Web page
(3) When logged select a site **CSCI835 (JIS20) Database Systems**
(4) Scroll down to a section **SUBMISSIONS**
(5) Click at a link **In this place you can submit the outcomes of Assignment 2**
(6) Click at a button **Add Submission**
(7) Move a file **solution1.lst** into an area **You can drag and drop files here to add them**. You can also use a link **Add**...
(8) Repeat a step (7) for the files **solution2.lst**, and **solution3.lst**.
(9) Click at a button **Save changes**
(10) Click at a button **Submit assignment**
(11) Click at the checkbox with a text attached: **By checking this box, I confirm that this submission is my own work,** ... in order to confirm the authorship of your submission.
(12) Click at a button **Continue**

End of specification