HW Assignment #7

Oracle PL/SQL

stored functions/procedures, constraints/triggers

KAIST

Prof. Myoung Ho Kim

Contents

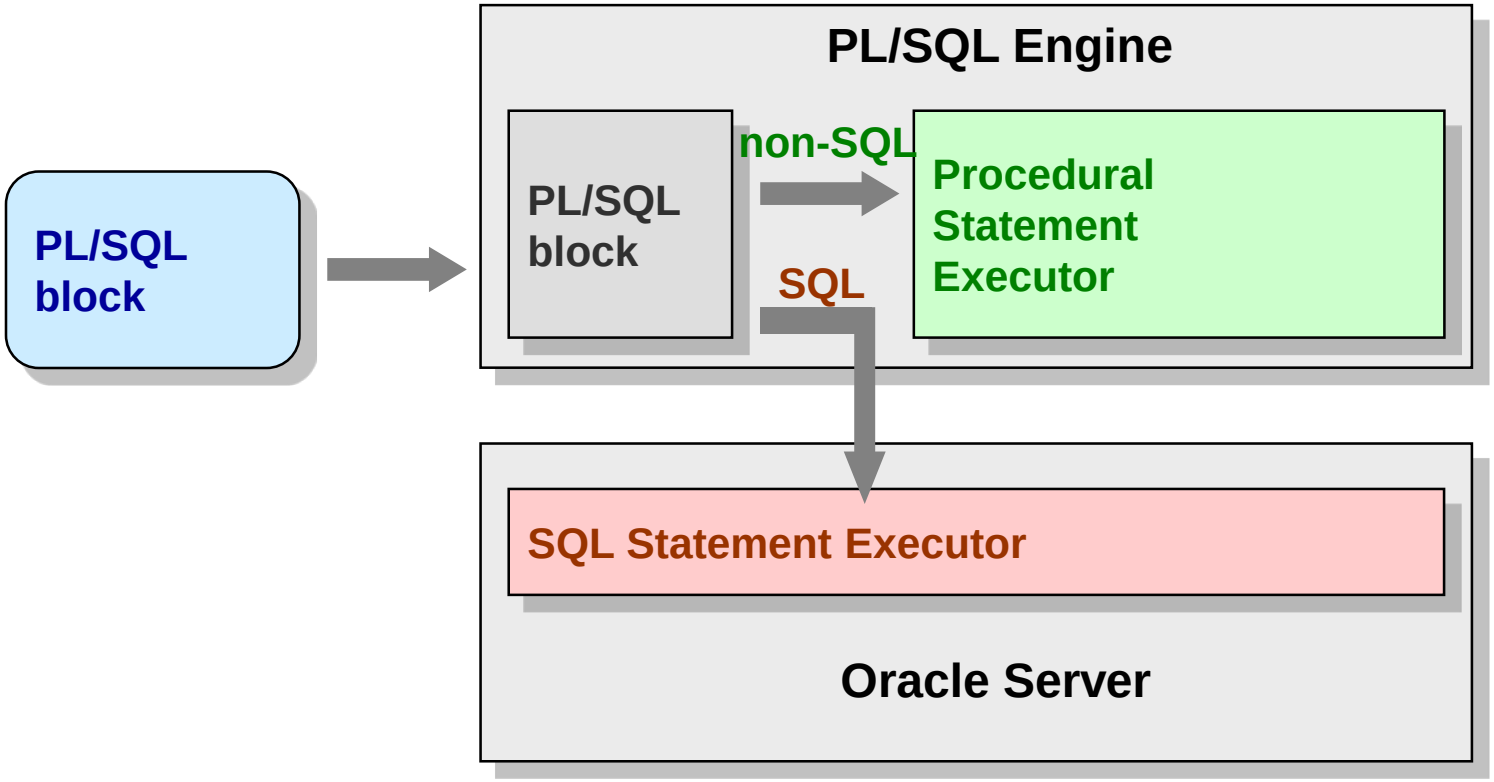
- PL/SQL
  - Introduction to PL/SQL
  - Stored procedures/functions
  - Variables
  - Control structures
  - Cursors
  - Exception handling
  - Triggers
- Calling stored procedures/functions in PHP
- Homework Assignment #7

PL/SQL: Introduction

- PL/SQL (Procedural Language-SQL)
  - Procedural SQL Language features
    - » Modularizing the program development
    - » Variable declaration
    - » Loop statements and conditional statement
    - » Cursor
    - » Support exception handling
    - » Procedures, functions
  - Enhancing the application performance
    - » Many logics or functions can be implemented in databases

PL/SQL: Introduction

- PL/SQL environment



PL/SQL: Introduction

- PL/SQL Block Structure

```
Header

IS
 -- declarations        Declaration Section
BEGIN
-- statements           Execution Section
        EXCEPTION
                        Exception Section
END;
```

PL/SQL: Introduction

- PL/SQL subprogram
  - ***Named PL/SQL block***
    - » Stored procedures and stored functions
      - stored in database and can be called repeatedly
      - functions return a result
    - » Triggers
      - stored subprogram associated with a table, view, or event
      - invoked when specific events occur
        - e.g. INSERT, DELETE, UPDATE

PL/SQL: Stored Procedures

- Creation

```
CREATE [OR REPLACE] PROCEDURE procedure_name
[(argument1 [mode] data_type1,
  argument2 [mode] data_type2,
  ......)]
IS
(variable declarations)
BEGIN
(code for execution)
[EXCEPTION]
(exception handling)
END;
```

mode = [IN|OUT|INOUT]

- Deletion

```
DROP PROCEDURE procedure_name;
```

(EX) Stored Procedure

procedure.sql

```
CREATE OR REPLACE PROCEDURE update_grade
(v_sid IN NUMBER)

IS

BEGIN
    UPDATE ScoreRecord
    SET Score= Score + 0.3
    WHERE Studentid = v_sid AND
          courseID = 'CS360';
END;
/
```

'/' in the last line means "execute CREATE statement"

Execute in SQL*Plus

```
SQL> @procedure.sql

Procedure created.

SQL> execute update_grade(70541);

PL/SQL procedure successfully completed.

SQL>
```

To see compile errors
Type ***SHOW ERRORS***
in SQL-Plus

PL/SQL: Stored Procedures

- How to execute stored procedures
  - In SQL*Plus: use EXECUTE statement

```
SQL> EXECUTE update_grade( 90 );
```

  - In PL/SQL block: write the procedure's name to be called within BEGIN, END clause

```
BEGIN
   update_grade( 90 );
END;
```

"CALL" is not needed

If cannot execute the stored procedure in console,
You type "**show serveroutput**" and "**set serveroutput on**"
when the status is off

PL/SQL: Stored Functions

- Creation

```
CREATE [OR REPLACE] FUNCTION function_name
[(argument1 [mode] data_type1,
  argument2 [mode] data_type2,
  ......)]
RETURN data_type
IS
(variable declarations)
BEGIN
(code for execution)
RETURN (value);
[EXCEPTION]
(exception handling)
END;
```

- Deletion

```
DROP FUNCTION function_name;
```

PL/SQL: Example - stored function

function.sql

```
CREATE OR REPLACE FUNCTION get_score
(v_sid IN NUMBER)
RETURN NUMBER
IS
   v_score NUMBER;
BEGIN
   SELECT Score INTO v_score
      FROM ScoreRecord
      WHERE StudentId = v_sid AND
               CourseID = 'CS360';
   RETURN v_score;
END;
/
```

'/' in the last line means "execute CREATE statement"

Execute in SQL*Plus

```
SQL> @function.sql;

Function created.

SQL> VAR score NUMBER;
SQL> EXECUTE :score := get_score(71041);

PL/SQL procedure successfully completed.

SQL> PRINT score;

     SCORE
----------
       3.7

SQL>
```

PL/SQL: Stored Functions

- How to execute
  - In SQL*Plus:
    - » Declare a bind variable to save the return value
    - » type EXECUTE statement(to see the return value, use PRINT statement)

```
SQL> VARIABLE score NUMBER;
SQL> EXECUTE :score := get_score(20160000);
```

  - In procedures:

```
score NUMBER;
BEGIN
   score := get_score(20160000);
END;
```

PL/SQL: Variables

- Declaration
  - Declare in IS

```
variable_name [CONSTANT] data_type [NOT NULL] [:=
value];
```

Example

```
v_empno NUMBER := 0;
```

- Assign a value

```
variable_name := value or expression;
```

Example

```
v_price := 5000;
tax := price * tax_rate;
amount := TO_NUMBER(SUBSTR('750 dollars', 1, 3));
```

PL/SQL: Control structures

- Conditional control (IF, END IF)

Syntax

```
IF condition1 THEN …
[ELSEIF condition2 THEN …]
[ELSE …]
END IF
```

Example

```
BEGIN
        IF sales > 50000 THEN
           bonus := 1500;
        ELSEIF sales > 35000 THEN
           bonus := 500;
        ELSE
           bonus := 100;
        END IF;
END;
```

PL/SQL: Control structures

Supplement
- maybe convenient if you use it

- Iterative control (LOOP, END LOOP)

Syntax

```
LOOP
        sequence of statements
        [EXIT WHEN condition]
END LOOP
```

Example

```
DECLARE
v_cnt NUMBER := 100;
BEGIN
LOOP
        INSERT INTO emp (empno, ename, hiredate)
             VALUES (v_cnt , 'Tom', sysdate);
        v_cnt := v_cnt + 1;
        EXIT WHEN v_cnt > 110; -> exit condition
END LOOP;
END;
/
```

PL/SQL: Control structures

Supplement
- maybe convenient if you use it

- Iterative control (FOR, END LOOP)

Syntax

```
FOR counter IN [REVERSE] min_value .. max_value LOOP
   sequence_of_statements
END LOOP
```

Example

```
DECLARE
i NUMBER;
BEGIN
FOR i IN 1 .. 100 LOOP
        UPDATE sales SET custno = customer_id
        WHERE serial_num = serial_num_seq;
END LOOP;
END;
```

PL/SQL: Control structures

Supplement
- maybe convenient if you use it

- Iterative control (WHILE, END LOOP)

Syntax

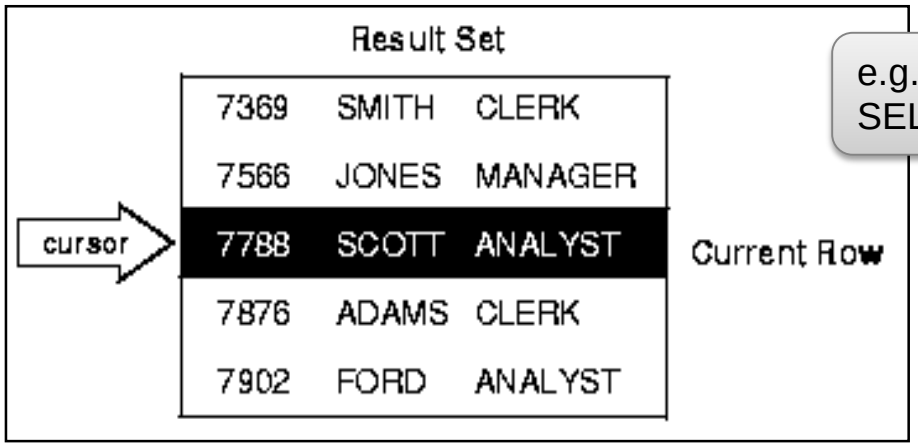
```
WHILE condition LOOP
   sequence_of_statements
END LOOP
```

Example

```
WHILE total < 25000 LOOP
        ...
        SELECT sal INTO salary FROM emp WHERE ...
        total := total + salary;
END LOOP;
```

PL/SQL: Cursors

- Cursor
  - Used to access the data in the workspace



e.g. access the result tuples of SELECT statement

PL/SQL: For Loop Cursors

- Named cursors
  - Declared inside BEGIN block;

```
FOR cursor_name IN (SQL statement) LOOP
        --do something..
END LOOP;
```

- Example

```
CREATE OR REPLACE PROCEDURE emp_process
IS
BEGIN
   FOR emp_cursor IN (SELECT empno, ename, sal
                      FROM emp
                      WHERE deptno = 20 )
   LOOP
        DBMS_OUTPUT.PUTLINE(emp_cursor.empno ||
                 emp_cursor.ename || emp_cursor.sal);
   END LOOP;
END;
/
```

"**show serveroutput**"

PL/SQL: Exceptions

- Exceptions
  - In PL/SQL an error condition is called “exception”
    - » e.g. divided by zero, storage permission denied
  - When an error occurs, an exception is raised
    - » i.e. normal execution stops and control transfers to the exception-handling part of the PL/SQL block
  - When is an exception raised?
    - » System violation occurs
    - » PL/SQL code calls “RAISE” statement

PL/SQL: Predefined Exceptions

Supplement
- maybe convenient if you use it

- **Predefined PL/SQL exceptions**
  - **named and unnamed exceptions**
    - » 21 system errors have predefined names
    - » other exceptions have their error code but have no name

Exception names	Description
ZERO_DEVIDE	Attempts to divide a number by 0
DUP_VAL_ON_INDEX	Attempts to store duplicate values in a column has UNIQUE constraint
NOT_LOGGED_ON	Issues database call without being connected to Oracle database
…	…

PL/SQL: User Defined Exceptions

- Declaration
  - Declare in IS SECTION

```
exception_name EXCEPTION;
```

- Usage
  - Raise an exception with “RAISE” statement in BEGIN SECTION

```
RAISE exception_name;
```

PL/SQL: Exception Handling

- EXCEPTION SECTION

OTHERS handler catches all exceptions that the block does not name specifically

```
EXCEPTION
   WHEN exception_name1 [OR exception_name2 ...] THEN
      sequence_of_statements;
      ......
   [WHEN exception_name3 [OR exception_name4 ...] THEN
      sequence_of_statements; ......]
   [WHEN OTHERS THEN
      sequence_of_statements; ......]
```

  - RAISE_APPLICATION_ERROR(*error_number*, *message*)
    - » Define user's own error message that printed in stdout
    - » *error_number* a negative integer in the range -20000 ~ -20999 and *message* is a character string up to 2048 bytes long

"Please type valid phone number" is more meaningful rather than "ORA-02290: Check constraint violation error"

(EX) Exception Handling

```
CREATE OR REPLACE PROCEDURE User_Exception
(v_deptno IN emp.deptno%type)
IS
        user_defined_error EXCEPTION;
        cnt NUMBER;
BEGIN
        SELECT COUNT(empno) INTO cnt
           FROM emp WHERE deptno = v_deptno;
        IF cnt < 5 THEN
           RAISE user_defined_error;
        ENDIF;
EXCEPTION
        WHEN DUP_VAL_ON_INDEX THEN
           RAISE_APPLICATION_ERROR(-20000, 'It is already in the table'');
        WHEN user_defined_error THEN
           RAISE_APPLICATION_ERROR(-20001, 'The department has too low employees');
END;
```

PL/SQL: Exception Handling

Supplement
- maybe convenient if you use it

- Handling unnamed exceptions
  - » exceptions that have only system error codes
  - » e.g. CHECK constraint violation

  1. Use OTHERS handler
  2. Give a name to the exception

- Give names to the unnamed exceptions
  - Declare user-defined exceptions
  - Assign the exception names to the unnamed error numbers
    - » PRAGMA EXCEPTION_INIT(*user-defined exception*, *error_number*)

Error number that returned by the system

(EX) Handling Unnamed Exception

Supplement
- maybe convenient if you use it

ORA-02290: Check constraint violation error

```
CREATE OR REPLACE PROCEDURE Exception_Naming
IS
    check_violated EXCEPTION;
    PRAGMA EXCEPTION_INIT(check_violated, -2290);
BEGIN
    ...
EXCEPTION
    WHEN check_violated THEN
        RAISE_APPLICATION_ERROR(-20001, 'It is invalid value');
END;
```

PL/SQL: Triggers

- Event-driven PL/SQL subprogram
  - Database calls a trigger when a specific DML(data manipulation language) on a table is executed

- Elements
  - TIMING
    - » BEFORE, AFTER: when to execute the trigger
  - TRIGGER_EVENT
    - » INSERT, UPDATE, DELETE: execute the trigger when INSERT, UPDATE, DELETE events occur
  - LEVEL
    - » STATEMENT: execute the trigger once
    - » ROW: execute the trigger for each row

PL/SQL: Triggers

- Creation

```
CREATE [OR REPLACE] TRIGGER trigger_name
TIMING TRIGGER_EVENT1 [OR TRIGGER_EVENT2 ...]
[OF column_name] ON table_name
[REFERENCING
[NEW AS new_row_name][OLD AS old_row_name]]
[FOR EACH ROW] [WHEN (condition)]
DECLARE
(variable declarations)
BEGIN
(PL/SQL code for execution)
END;
```

TIMING = BEFORE | AFTER

$TRIGGER\_EVENT_x$ = INSERT | UPDATE | DELETE

- Deletion

```
DROP TRIGGER trigger_name;
```

(Ex) Triggers

```
CREATE OR REPLACE TRIGGER secure_emp
BEFORE INSERT OR UPDATE OR DELETE ON s_emp
BEGIN
        IF (TO_CHAR(SYSDATE, 'DY') IN ('SAT', 'SUN')) OR
           (TO_CHAR(SYSDATE, 'HH24') NOT BETWEEN '09' AND '16')
        THEN
           RAISE_APPLICATION_ERROR(-20201, 'Unavailable time');
        END IF;
END;
```

```
CREATE OR REPLACE TRIGGER prod_update
AFTER UPDATE OF dscp ON product
FOR EACH ROW
WHEN new.price < old.price * 1.5
BEGIN
        UPDATE order_details
        SET p_dscp = :old.dscp
        WHERE p_id = :old.p_id;
END;
```

PL/SQL: Triggers

- Variable for referring events
  - FOR EACH ROW
    - » In BEGIN SECTION
      - :old – the row before being updated
      - :new – the row after being update
    - » In WHEN condition
      - Use old, new (no preceding colon!!)

❖ The differences between SQL standard and Oracle triggers (refer to the following URL)

» http://www-db.stanford.edu/~ullman/fcdb/oracle/or-triggers.html

Execute a Stored Procedure in PHP

- In PEAR DB,
  - Stored procedures can be called by using BEGIN END statement
    - » Example

```
//PEAR DB
$conn = DB::connect('oci8://s20160000:s20160000@dbclick.kaist.ac.kr'
                                    .':1521/orcl');
$stmt = $conn->prepare("begin update_sal(?); end;");
$bindvars = array(30);
$res = $conn->execute($stmt,$bindvars);
...
```

Execute a Stored Function in PHP

- In PEAR DB,
  - Stored functions can be called by using SELECT statement
    - » Example

```
//PEAR DB
$conn = DB::connect('oci8://s20160000:s20160000@dbclick.kaist.ac.kr'
                                    .':1521/orcl');
$stmt = $conn->prepare("select get_sal(?) from dual");
$bindvars = array(30);
$res = $conn->execute($stmt,$bindvars);
$sal = $res->fetchRow()[0]; //get the result of get_sal()
```

(EX) Get Error Code in PHP

```
$stmt = $conn->prepare("begin update_sal(?); end;");
$bindvars = array(30);
$res = $conn->execute($stmt,$bindvars);
if(DB::isError($res)){
        echo 'Standard Message: ' . $conn->getMessage() . "<br/>";
        echo 'Standard Code: ' . $conn->getCode() . "<br/>";
        echo 'DBMS/User Message: ' . $conn->getUserInfo() . "<br/>";
        echo 'DBMS/Debug Message: ' . $conn->getDebugInfo() . "<br/>";
}
```

**User error codes can be caught by using getUserInfo()*

DBMS/User Message: [nativecode=ORA-20000: unavailable dates]…

Homework Assignment #7

Prerequirements

- Building web pages
  1. Download *HW7web.zip* from KLMS and unzip
  2. Copy unzipped files to
     (directory that Apache is installed)/htdocs
     - » Access to http://localhost/index.php and check

- SQL files
  - Download *HW7sqls.zip* from KLMS and upzip
    - » 4 files: *HW7db.sql, problem2.sql, problem3.sql, problem4.sql*
  - If you want to execute the sql files, copy unzipped files to
    (directory that Oracle Client is installed)\BIN

Prerequirements (cont'd)

- Creating a DB connection to CS360 Oracle server
  - Open ***db.connect.php*** file in *Config* folder
  - Edit constants, **DB_USER** and **DB_PASSWORD**

http://localhost/index.php

CS360 HW#7

Insert a new PC

Find the center price

TABLE : product

no.	MAKER	MODEL	TYPE
1	A	1001	pc
2	A	1002	pc
3	A	1003	pc
4	B	1004	pc
5	B	1005	pc
6	B	1006	pc

TABLE : pc

no.	MODEL	SPEED	RAM	HD	PRICE
1	1001	2.66	1024	250	2000
2	1002	2.1	512	250	995

HW7 problems

- Product DB schema as used in homework 4
  - » Primary key is underlined
  - » Reference keys are italic
  - – PRODUCT(maker, **<u>model</u>**, type)
  - – PC(***model***, speed, ram, hd, price)
  - – LAPTOP(***model***, speed, ram, hd, screen, price)
  - – PRINTER(***model***, color, type, price)

- Total 5 problems

HW7 problems (cont'd)

- Problem 1 (20 points): Table creation with constraints
  - Open *HW7db.sql* and add the following constraints
    - » Data constraints in PRODUCT table
      - maker: **NULL is not allowed**
      - type: **only 'pc', 'laptop', or 'printer'**
  - RUN *HW7db.sql* in *SQLPLUS*
    - » SQLPLUS Command: @HW7db.sql

HW7 problems (cont'd)

- Problem 2 (20 points)
  - Make a **trigger**, *CheckNumProduct*
    - » Before inserting a new product into PRODUCT table,
      if the number of products of a certain manufacturer is larger than 9,
      then raise the application error with error code -20000
      - It means that for each manufacturer, the maximum total number of products is 10
      - You can assume that for each model there is either one computer or one printer
  - The trigger *CheckNumProduct* should be implemented in *problem2.sql* file

HW7 problems (cont'd)

- Problem 3 (20 points)
  - Make a **stored procedure,** *insert_pc* that insert a new computer information into both PRODUCT and PC tables
    - » Functions
      - Input parameter: manufacturer, model number, speed, ram, hard disk size and price of a new computer
      - After inserting into PRODUCT table, insert the new computer into PC table
    - » Exception
      - When DUP_VAL_ON_INDEX error occurs, raise application error with error code -20001
  - The stored procedure *insert_pc* should be implemented in *problem3.sql* file

HW7 problems (cont'd)

- Problem 4 (20 points)
  - Make a **stored function**, *findCenterPrice*, that finds the most center price in PC table
    - » Functions
      - **No input parameter**
      - Return value: the most center price in PC
        - The most center price

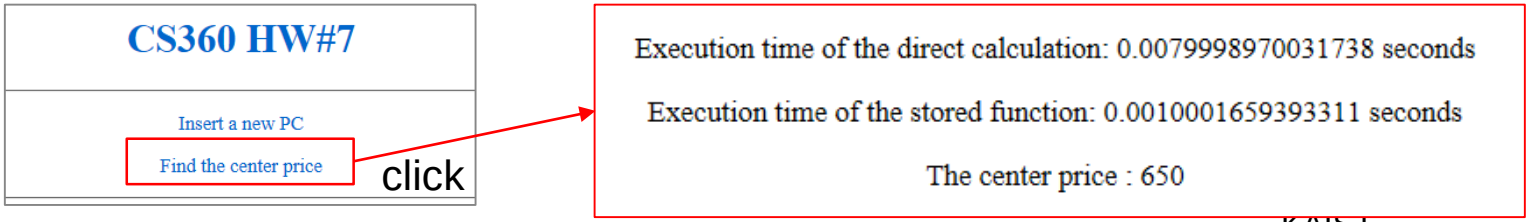
          Let $\{p_1, p_2, \ldots, p_n\}$ is a bag of prices in PC table

          If $p_k \in \{p_1, p_2, \ldots, p_n\}$ is the most center price, then $\sum_{i=1}^{n} |p_k - p_i|$ has the minimum
  - The stored function *findCenterPrice* should be implemented in *problem4.sql* file

HW7 problems (cont'd)

- Problem 5 (20 points)
  - Make a **PHP function**, *cal_center_price* that finds the most center price in PC table
    - DO NOT CALL stored functions in *cal_center_price*
    - Calculate the most center price by bring data from the database
  - The PHP function *cal_center_price* should be implemented in *findCenterPrice.php* file in *problems* folder of HW7Web.zip

❖ You can compare the execution time of calling stored function *findCenterPrice* with that of calling PHP function *cal_center_price* through 'Find the center price' menu in *index.php*



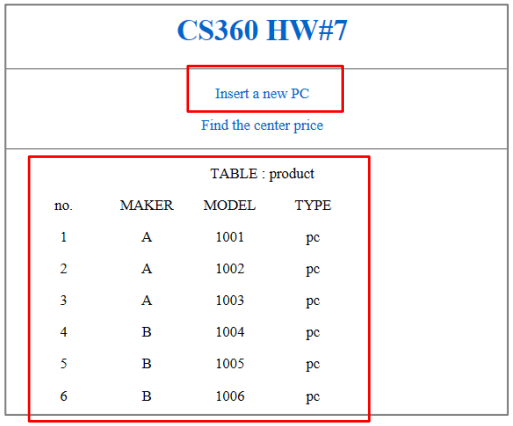
HW7 Submission

- Files to submit
  - 1. Total 5 files with 5 problems
    - » ***HW7db.sql, problem2.sql, problem3.sql, problem4.sql***
    - » ***findCenterPrice.php*** in the ‘problems’ folder
  - 2. Archive them into [student ID].zip and upload it to KLMS
    - » Ex) 20160000.zip, not [20160000].zip

- Due date
  - **November 30(Wed), 2:00 a.m.**
    - » No delay
    - » No copy (zero score for each)

HW7 Noted items

- Given default functions
  - Lists of PRODUCT and PC tables in index.php
  - Insertion into PC table
    - » **When stored procedure 'insert_pc' is implemented**



- TA info
  - Kwang Hee, Lee (kwanghee@dbserver.kaist.ac.kr)

- Office hour
  - Room#404, N1 building
  - Wed: 4:00~5:30, Fri: 2:30~4:00

Reference

- **Text book (**Database systems - the complete book 2nd edition)
  - Chap 6. SQL
  - Chap 7. Constraints and Triggers
  - Chap 9.4 Stored procedures

- **PL/SQL User's Guide and Reference**
  - http://docs.oracle.com/cd/E11882_01/appdev.112/e25519/toc.htm

- Oracle PL/SQL tutorial
  - http://www.plsqltutorial.com/what-is-plsql/