

Abgabetermin: 14.11.2018, 13:30 Uhr

☒ DES31UE Niklas Name Papesh Konstantin Aufwand in h 8
☐ DES32UE Niklas
☐ DES33UE Traxler Punkte _____ Kurzzeichen Tutor _____

Ziel dieser Übung ist die Einführung in die Grundlagen von PL/SQL und die Erstellung von gespeicherten Prozeduren in der Datenbank. Der Unterschied zwischen SQL und PL/SQL und ausgewählte Details werden in einem Theorie-Block recherchiert.

Zusätzliche Hinweise

Fügen Sie für jedes Beispiel (auch Unterpunkte) den entsprechenden PL/SQL-Code in ihr Abgabedokument ein. Geben Sie also auch Zwischenergebnisse ab und kennzeichnen Sie die Ausarbeitung der jeweiligen Aufgabe.

1. PL/SQL Grundlagen (8 Punkte)

1. Führen Sie folgendes Skript UE06_01_01.sql aus, um die Tabelle top_salaries zu erstellen, in der die Gehälter der Angestellten gespeichert werden sollen.

```
DROP TABLE top_salaries;  
CREATE TABLE top_salaries (salary NUMBER(8,2));
```

Skript UE06_01_01.sql

2. Machen Sie sich mit nachfolgendem Skript UE06_01_02.sql vertraut. Verwenden Sie ggf. die Oracle Referenz (PL/SQL User's Guide and Reference), um Befehle nachzulesen. Welche Anweisungen sind SQL- bzw. PL/SQL-Kommandos?

```
DELETE FROM top_salaries;  
DECLARE  
    num    NUMBER(3) := &p_num;  
    sal     employees.salary%TYPE;  
    CURSOR emp_cursor IS  
        SELECT DISTINCT salary  
        FROM employees  
        ORDER BY salary DESC;  
BEGIN  
    OPEN emp_cursor;  
    FETCH emp_cursor INTO sal;  
    WHILE emp_cursor%ROWCOUNT <= num AND emp_cursor%FOUND LOOP  
        INSERT INTO top_salaries (salary)  
        VALUES (sal);  
        FETCH emp_cursor INTO sal;  
    END LOOP;  
    CLOSE emp_cursor;  
END;  
/  
SELECT * FROM top_salaries;
```

Skript UE06_01_02.sql

3. Testen Sie verschiedene Spezialfälle, zum Beispiel wenn $n = 0$ oder wenn n größer als die Zahl der Angestellten in der Tabelle employees ist. Kommentieren Sie Ihre Tests.

4. Zusätzlich zum Gehalt soll auch die Anzahl der Mitarbeiter abgespeichert werden, die dieses Gehalt verdienen. Erweitern Sie die Tabelle `top_salaries` um das Feld `emp_cnt` und wählen Sie einen passenden Datentyp aus. Definieren Sie einen Primärschlüssel und ein Check Constraint zur Sicherstellung dass `emp_cnt` größer als Null ist. Speichern Sie das DDL-Skript ab.
5. Modifizieren Sie das Skript `UE06_01_02.sql`, um das neue Feld korrekt zu befüllen. Speichern Sie das modifizierte Skript ab.

2. PL/SQL Prozeduren

(6 Punkte)

1. Für die Datensätze in der Tabelle `top_salaries` werden Logging-Daten von der Erstellung sowie von der letzten Änderung benötigt. Erweitern Sie dazu die Tabelle `top_salaries` um die Felder `createdBy`, `dateCreated`, `modifiedBy` und `dateModified`. Bei der Anlage eines Datensatzes sind die `Created`- und `Modifed`-Felder ident. Speichern Sie das DDL-Skript ab.
2. Erstellen Sie eine Datenbank-Prozedur `InsertTopSalaries`, die einen Datensatz in der Tabelle `top_salaries` anlegt und die Logging-Felder befüllt. Für die Logging-Felder verwenden Sie die Systemfunktionen `USER` und `SYSDATE`. Die Systemfunktion `USER` liefert den Namen des angemeldeten Benutzers. Die Systemfunktion `SYSDATE` liefert das aktuelle Systemdatum. Das Skript für die Erstellung der Prozedur speichern Sie ab. Die Prozedur soll folgende Spezifikation aufweisen:

```
CREATE OR REPLACE PROCEDURE InsertTopSalaries (  
    pSalary          IN NUMBER,  
    pEmp_cnt         IN NUMBER)  
IS  
    . . .
```

3. Ersetzen Sie die `INSERT`-Anweisung im Skript `UE06_01_02` durch die in der vorherigen Aufgabe erstellten Prozedur `InsertTopSalaries` und überprüfen Sie das Ergebnis. Speichern Sie das Skript ab. Hinweis: Um auch die Uhrzeit zu sehen, können Sie das Datumsformat mit folgendem Kommando festlegen:

```
ALTER SESSION SET NLS_DATE_FORMAT = 'dd.mm.yyyy hh24:mi:ss';
```

3. Performance-Optimierung

(5 Punkte - 3 + 1 + 1)

Erstellen Sie sich mit dem gegebenen Skript `UE06_03_01.sql` eine Tabelle `my_payment` indem Sie die Datensätze der Tabelle `payment` einfügen. Fügen Sie eine weitere Spalte `penalty` der Tabelle hinzu. Ermitteln Sie nun für jeden Bezahlvorgang (der einem Verleihvorgang entspricht) ob der Film länger verliehen war, als unter `rental_duration` angegeben. Das gegebene Skript enthält einen anonymen Block, der diese Berechnung in einer Schleife durchführt. Führen Sie diesen Block aus und notieren Sie die ermittelte Laufzeit.

1. Entwickeln Sie eine weitere Version des Skripts und eliminieren Sie die Schleife. Führen Sie also in einem weiteren anonymen Block ein einfaches Update-Statement aus, das die gleiche Berechnung vornimmt.
2. Stellen Sie sicher, dass die Version mit der Schleife und Ihre Version mit dem einzelnen Update-Statement die gleichen Werte berechnen.
3. Führen Sie eine Zeitmessung durch und interpretieren Sie das Ergebnis. Löschen Sie die Tabelle(n) wieder.

4.

```
CREATE TABLE my_payment AS
SELECT *
FROM payment
WHERE rental_id IS NOT NULL;
ALTER TABLE my_payment ADD PRIMARY KEY (payment_id);
ALTER TABLE my_payment ADD penalty NUMBER;

-- UPDATE in loop
DECLARE
    starttime NUMBER;
    total NUMBER;
    maxRent NUMBER := 0;
    actualRent NUMBER := 0;
BEGIN
    starttime := DBMS_UTILITY.GET_TIME();
    FOR mp IN (SELECT amount, rental_id, payment_id, payment_date FROM my_payment) LOOP
        SELECT MAX(rental_duration) INTO maxRent
        FROM film
        INNER JOIN inventory USING (film_id)
        INNER JOIN rental USING (inventory_id) WHERE rental_id = mp.rental_id;

        SELECT MAX(CEIL(return_date - rental_date)) INTO actualRent
        FROM rental
        WHERE rental_id = mp.rental_id;

        IF actualRent > maxRent THEN
            UPDATE my_payment
            SET penalty = amount * 1.15
            WHERE mp.payment_id = payment_id;
        END IF;

    END LOOP;
    total := DBMS_UTILITY.GET_TIME() - starttime;
    DBMS_OUTPUT.PUT_LINE('PL/SQL LOOP: ' || total / 100 || ' seconds');
END;
/

DROP TABLE my_payment;
```

Skript UE06_03_01.sql

4. Multiple Choice

(5 Punkte – 1+1+1+2)

Wählen Sie aus den gegebenen Antworten die richtigen aus. Im Zweifelsfall begründen Sie.

1. PL/SQL eignet sich gut um

- ☐ DDL-Anweisungen kompakt auszuführen.
- ☒ SQL-Anweisungen in Verbindung mit Schleifen und Bedingungen auszuführen.
- ☒ wiederkehrende Aufgaben auszuführen.
- ☐ SQL-Anweisungen effizient auszuführen.

2. In PL/SQL

- ☐ dürfen Variablen nicht die gleichen Namen besitzen wie Tabellen oder Spalten.
- ☐ kann von der Tabelle dual nicht selektiert werden.
- ☒ sind SQL-Funktionen (zB Datum) ebenfalls verfügbar.
- ☒ darf kein COMMIT ausgeführt werden.

3. Wenn SQL-Anweisungen in einem PL/SQL-Block verwendet werden

- ☐ müssen diese extra als SQL gekennzeichnet werden.
- ☒ sind spezielle Schlüsselwörter (INTO, ...) für die Speicherung eines Ergebnisses notwendig.
- ☐ muss das Ergebnis aus einer Pseudo-Variable extrahiert werden.
- ☒ können diese mit anderen PL/SQL-Konstrukten gemischt werden.

4. Welche Aussagen sind wahr?

- ☒ Liefert ein SQL-Statement mehrere Ergebniszeilen, ist ein Cursor notwendig.
- ☐ Eine Variable kann auch als „NOT NULL“ deklariert werden.
- ☐ Hierarchische Abfragen (Rekursion) sind in PL/SQL nicht möglich.
- ☐ Eine Prozedur darf nur eine BEGIN- und eine END-Anweisung enthalten.
- ☒ Mit PL/SQL soll möglichst viele Business-Logik in die Datenbank gebracht werden.
- ☐ PL/SQL kann auch Java-Code ausführen. [PL/SQL an sich nicht, Oracle unterstützt PL/Java](#)
- ☒ IN- und OUT-Parameter einer Prozedur können einen Default-Wert besitzen.
- ☐ Ruft eine SQL-Anweisung eine Funktion auf, so darf diese keine DML-Inhalte besitzen.

1.1

```
[2018-11-13 12:54:02] Run /home/khp/git/fh-hgb/ws18/dem/ex06/UE06_01_01.sql
[2018-11-13 12:54:02] Connecting to Hagenberg...
[2018-11-13 12:54:03] Using batch mode (1000 insert/update/delete statements
max)
DROP TABLE top_salaries
[2018-11-13 12:54:03] 0 row(s) affected in 61 ms
CREATE TABLE top_salaries (salary NUMBER(8,2))
[2018-11-13 12:54:03] 0 row(s) affected in 35 ms
[2018-11-13 12:54:03] Summary: 2 of 2 statements executed in 474 ms (70 symbols
in file)
```

1.2


```
DELETE FROM top_salaries;
DECLARE
  num NUMBER(3) := &p_num;
  sal employees.salary%TYPE;
  CURSOR emp_cursor IS
    SELECT DISTINCT salary
    FROM employees
    ORDER BY salary DESC;
BEGIN
  OPEN emp_cursor;
  FETCH emp_cursor INTO sal;
  WHILE emp_cursor%ROWCOUNT <= num AND emp_cursor%FOUND LOOP
    INSERT INTO top_salaries (salary)
    VALUES (sal);
    FETCH emp_cursor INTO sal;
  END LOOP;
  CLOSE emp_cursor;
END;
/
SELECT * FROM top_salaries;
```

Grün: SQL

Gelb: PL/SQL

1.3

$n=0$

 salary ▼ 1

$n=9990$


```
[2018-11-13 13:06:35] [65000][6502] ORA-06502: PL/SQL: numeric or value error:
number precision too large
```

[2018-11-13 13:06:35] ORA-06512: at line 2

$n=M$

```
[2018-11-13 13:07:07] [65000][6550] ORA-06550: line 2, column 22:
[2018-11-13 13:07:07] PLS-00201: identifier 'M' must be declared
[2018-11-13 13:07:07] ORA-06550: line 2, column 9:
[2018-11-13 13:07:07] PL/SQL: Item ignored
[2018-11-13 13:07:07] ORA-06550: line 11, column 33:
[2018-11-13 13:07:07] PLS-00320: the declaration of the type of this expression
is incomplete or malformed
[2018-11-13 13:07:07] ORA-06550: line 11, column 4:
[2018-11-13 13:07:07] PL/SQL: Statement ignored
```

$n=-1$

 salary ▾ 1

1.4

```
ALTER TABLE top_salaries ADD (emp_cnt NUMBER(5), PRIMARY KEY (salary), CONSTRAINT age_gt_0 CHECK
(emp_cnt >= 0));
```

1.5

```
DELETE FROM top_salaries;
DECLARE
    num NUMBER(3) := &p_num;
    sal employees.salary%TYPE;
    c_emp_cnt INTEGER;
    CURSOR emp_cursor IS
        SELECT salary, COUNT(salary) AS sal_count
        FROM employees
        GROUP BY salary
        ORDER BY salary DESC;
    rec emp_cursor%ROWTYPE;
BEGIN
    OPEN emp_cursor;
    FETCH emp_cursor INTO rec;
    WHILE emp_cursor%ROWCOUNT <= num AND emp_cursor%FOUND LOOP
        INSERT INTO top_salaries (salary, emp_cnt)
        VALUES (rec.salary, rec.sal_count);
        FETCH emp_cursor INTO rec;
    END LOOP;
    CLOSE emp_cursor;
END;
/
```

2.1

```
ALTER TABLE top_salaries ADD (createdBy VARCHAR(30), dateCreated TIMESTAMP, modifiedBy
VARCHAR(30), dateModified TIMESTAMP);
```

2.2

```
CREATE OR REPLACE PROCEDURE InsertTopSalaries (
pSalary IN NUMBER,
pEmp_cnt IN NUMBER)
IS
BEGIN
    INSERT INTO top_salaries VALUES (pSalary, pEmp_cnt, USER, SYSDATE, USER, SYSDATE);
END;
```

2.3

```
DELETE FROM top_salaries;
DECLARE
    num NUMBER(3) := &p_num;
    sal employees.salary%TYPE;
    c_emp_cnt INTEGER;
    CURSOR emp_cursor IS
        SELECT salary, COUNT(salary) AS sal_count
        FROM employees
        GROUP BY salary
        ORDER BY salary DESC;
    rec emp_cursor%ROWTYPE;
BEGIN
    OPEN emp_cursor;
    FETCH emp_cursor INTO rec;
    WHILE emp_cursor%ROWCOUNT <= num AND emp_cursor%FOUND LOOP
        InsertTopSalaries(rec.salary, rec.sal_count);
    END LOOP;
    CLOSE emp_cursor;
END;
/
```

3.1

```
DECLARE
    starttime NUMBER;
    total NUMBER;
    amount my_payment.amount%TYPE;
    rental_id my_payment.rental_id%TYPE;
    payment_id my_payment.payment_id%TYPE;
    payment_date my_payment.payment_date%TYPE;
    maxrent NUMBER := 0;
    actualrent NUMBER := 0;
    CURSOR mp IS
        SELECT amount, rental_id, payment_id, payment_date, CEIL(return_date - rental_date),
        rental_duration
        FROM my_payment
        INNER JOIN rental USING (rental_id)
        INNER JOIN inventory USING (inventory_id)
        INNER JOIN film USING (film_id);
BEGIN
```

```

starttime := dbms_utility.GET_TIME();
UPDATE (SELECT my_payment.penalty, amount, CEIL(return_date - rental_date) AS actualrent,
rental_duration AS maxrent
      FROM my_payment
            INNER JOIN rental USING (rental_id)
            INNER JOIN inventory USING (inventory_id)
            INNER JOIN film USING (film_id))
      mp
SET mp.penalty = amount * 1.15 WHERE actualrent > maxrent;
total := dbms_utility.GET_TIME() - starttime;
dbms_output.PUT_LINE('UPDATE: ' || total / 100 || ' seconds');
END;
/

```

3.2

Schleife:

1	188	7	1	5441	13.95	2015-10-31 00:00:00	2015-11-06 07:12:32.000000	16.0425
2	189	7	1	5921	6.87	2015-11-02 00:00:00	2015-11-06 07:12:32.000000	<null>
3	375	14	1	9592	5.56	2014-11-16 00:00:00	2015-11-06 07:12:32.000000	<null>
4	376	14	1	10348	3.38	2014-02-22 00:00:00	2015-11-06 07:12:32.000000	<null>
5	377	14	2	10526	5.58	2015-08-09 00:00:00	2015-11-06 07:12:32.000000	<null>
6	378	14	1	11480	6.26	2014-01-01 00:00:00	2015-11-06 07:12:32.000000	7.199
7	379	14	2	11528	7.92	2015-02-15 00:00:00	2015-11-06 07:12:32.000000	9.108
8	380	14	1	12668	20.22	2014-10-14 00:00:00	2015-11-06 07:12:32.000000	23.253
9	381	14	1	13757	13.45	2014-06-05 00:00:00	2015-11-06 07:12:32.000000	<null>
10	382	14	5	15015	14.94	2014-07-15 00:00:00	2015-11-06 07:12:32.000000	17.181

Update:

1	188	7	1	5441	13.95	2015-10-31 00:00:00	2015-11-06 07:12:32.000000	16.0425
2	189	7	1	5921	6.87	2015-11-02 00:00:00	2015-11-06 07:12:32.000000	<null>
3	375	14	1	9592	5.56	2014-11-16 00:00:00	2015-11-06 07:12:32.000000	<null>
4	376	14	1	10348	3.38	2014-02-22 00:00:00	2015-11-06 07:12:32.000000	<null>
5	377	14	2	10526	5.58	2015-08-09 00:00:00	2015-11-06 07:12:32.000000	<null>
6	378	14	1	11480	6.26	2014-01-01 00:00:00	2015-11-06 07:12:32.000000	7.199
7	379	14	2	11528	7.92	2015-02-15 00:00:00	2015-11-06 07:12:32.000000	9.108
8	380	14	1	12668	20.22	2014-10-14 00:00:00	2015-11-06 07:12:32.000000	23.253
9	381	14	1	13757	13.45	2014-06-05 00:00:00	2015-11-06 07:12:32.000000	<null>
10	382	14	5	15015	14.94	2014-07-15 00:00:00	2015-11-06 07:12:32.000000	17.181

Ident

3.3

PL/SQL LOOP: completed in 10 s 373 ms

UPDATE: completed in 223 ms

Da die Schleife jedes Element einzeln fetchen, bearbeiten und zurückspeichern muss dauert sie um ein Vielfaches länger als das einfache Update-Statement. Auch werden insgesamt 3 Abfragen in der Schleife getätigt, was für die Laufzeit nicht von Vorteil ist. Daher sind Schleifen mit Abfragen wenn möglichst zu vermeiden in PL/SQL.