DES3UE Datenbanksysteme WS 2018 Übung 6

Abgabetermin: 14.11.2018, 13:30 Uhr

☐ **DES31UE Niklas** **Name** _____Niklas Vest_____ **Aufwand in h** ___3__
☐ **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.

```sql
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.
- 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.

Ausarbeitung UE06

PL/SQL Grundlagen

Anmerkung: Dieses Dokument ist grauenvoll formattiert. Ich bin vor kurzem auf Linux umgestiegen und bin noch nicht so vertraut mit den Werkzeugen und lerne noch bzw. suche nach Alternativen. Ich bitte um Entschuldigung.

1.3 Tests

	salary
1	24000.00
2	17000.00
3	13000.00
4	12000.00
5	11000.00
6	10500.00
7	9000.00
8	8600.00
9	8300.00
10	7000.00

Illustration 2: p_num = 10 führt zu 10 Tupeln.

salary

Illustration 1: p_num = 0 und p_num = -1 führen zu leerer Ergebnisrelation

	salary
1	24000.00
2	17000.00
3	13000.00
4	12000.00
5	11000.00
6	10500.00
7	9000.00
8	8600.00
9	8300.00
10	7000.00
11	6000.00
12	5800.00
13	4400.00
14	4200.00
15	3500.00
16	3100.00
17	2600.00
18	2500.00

Illustration 3: p_num = 30 führt zur Übernahme der der ganzen "Ergebnisrelation des Cursors".

1.4

```
DROP TABLE top_salaries;
ALTER TABLE top_salaries
    ADD (emp_cnt NUMBER DEFAULT 1 NOT NULL)
    ADD CONSTRAINT emp_cnt_gt_0 CHECK(emp_cnt > 0)
    ADD CONSTRAINT top_salaries_pk PRIMARY KEY(salary);
```

1.5

```
DECLARE
  -- employee
  num NUMBER(3) := &p_num;
  top_sal top_salaries%ROWTYPE;
  -- employee cursor, highest salary first
  CURSOR emp_cursor IS
    SELECT salary, COUNT(*)
    FROM employees
    GROUP BY salary
    ORDER BY salary DESC;
BEGIN
  OPEN emp_cursor;
  FETCH emp_cursor INTO top_sal;
  -- for as long as more rows should be read
  -- and while there are still more rows to process
  WHILE emp_cursor%ROWCOUNT <= num AND emp_cursor%FOUND LOOP
    -- add dat salary to the top_salaries table
    INSERT INTO top_salaries (salary, emp_cnt)
    VALUES (top_sal.salary, top_sal.emp_cnt);
    -- fetch next tuple
    FETCH emp_cursor INTO top_sal;
  END LOOP;
  CLOSE emp_cursor;
END;
/
```

	salary	emp_cnt
1	24000.00	1
2	17000.00	2
3	13000.00	1
4	12000.00	1
5	11000.00	1

Illustration 4: p_num = 5

PL/SQL Prozeduren

2.1

```
DELETE FROM top_salaries;
ALTER TABLE top_salaries
  ADD (created_by    VARCHAR2(50)  DEFAULT ''      NOT NULL)
  ADD (date_created  DATE           DEFAULT SYSDATE NOT NULL)
  ADD (modified_by   VARCHAR2(50)  DEFAULT ''      NOT NULL)
  ADD (date_modified DATE           DEFAULT SYSDATE NOT NULL);
DECLARE
  -- employee
  num NUMBER(3) := &p_num;
  -- salary
  sal employees.salary%TYPE;
  cnt top_salaries.emp_cnt%TYPE;
  -- employee cursor, highest salary first
```

```
  CURSOR emp_cursor IS
    SELECT salary, COUNT(*)
    FROM employees
    GROUP BY salary
    ORDER BY salary DESC;
BEGIN
  OPEN emp_cursor;
  FETCH emp_cursor INTO sal, cnt;
  -- for as long as more rows should be read
  -- and while there are still more rows to process
  WHILE emp_cursor%ROWCOUNT <= num AND emp_cursor%FOUND LOOP
    -- add dat salary to the top_salaries table
    INSERT INTO top_salaries (salary, emp_cnt, date_created, created_by, date_modified,
modified_by)
    VALUES (sal, cnt, SYSDATE, USER, SYSDATE, USER);
    -- fetch next tuple
    FETCH emp_cursor INTO sal, cnt;
  END LOOP;
  CLOSE emp_cursor;
END;
/
```

	salary	emp_cnt	created_by	date_created	modified_by	date_modified
1	24000.00	1	S1710307099	2018-11-13 21:59:41	S1710307099	2018-11-13 21:59:41
2	17000.00	2	S1710307099	2018-11-13 21:59:41	S1710307099	2018-11-13 21:59:41
3	13000.00	1	S1710307099	2018-11-13 21:59:41	S1710307099	2018-11-13 21:59:41
4	12000.00	1	S1710307099	2018-11-13 21:59:41	S1710307099	2018-11-13 21:59:41
5	11000.00	1	S1710307099	2018-11-13 21:59:41	S1710307099	2018-11-13 21:59:41
6	10500.00	1	S1710307099	2018-11-13 21:59:41	S1710307099	2018-11-13 21:59:41
7	9000.00	1	S1710307099	2018-11-13 21:59:41	S1710307099	2018-11-13 21:59:41
8	8600.00	1	S1710307099	2018-11-13 21:59:41	S1710307099	2018-11-13 21:59:41
9	8300.00	1	S1710307099	2018-11-13 21:59:41	S1710307099	2018-11-13 21:59:41
10	7000.00	1	S1710307099	2018-11-13 21:59:41	S1710307099	2018-11-13 21:59:41

Illustration 5: p_num = 10

2.2

```
CREATE OR REPLACE PROCEDURE InsertTopSalaries (pSalary IN NUMBER, pEmp_cnt IN NUMBER)
  IS
  BEGIN
    INSERT INTO top_salaries (salary, emp_cnt, date_created, created_by, date_modified,
modified_by)
    VALUES (psalary, pEmp_cnt, SYSDATE, USER, SYSDATE, USER);
  END;
  /
```

2.3

```
DECLARE
  -- employee
  num NUMBER(3) := &p_num;
  -- salary
  sal employees.salary%TYPE;
  cnt top_salaries.emp_cnt%TYPE;
  -- employee cursor, highest salary first
  CURSOR emp_cursor IS
    SELECT salary, COUNT(*)
    FROM employees
    GROUP BY salary
    ORDER BY salary DESC;
BEGIN
  OPEN emp_cursor;
```

```
  FETCH emp_cursor INTO sal, cnt;
  -- for as long as more rows should be read
  -- and while there are still more rows to process
  WHILE emp_cursor%ROWCOUNT <= num AND emp_cursor%FOUND LOOP
    -- add dat salary to the top_salaries table
    InsertTopSalaries(sal, cnt);
    -- fetch next tuple
    FETCH emp_cursor INTO sal, cnt;
  END LOOP;
  CLOSE emp_cursor;
END;
/
```

	salary	emp_cnt	created_by	date_created	modified_by	date_modified
1	24000.00	1	S1710307099	2018-11-13 22:05:27	S1710307099	2018-11-13 22:05:27
2	17000.00	2	S1710307099	2018-11-13 22:05:27	S1710307099	2018-11-13 22:05:27
3	13000.00	1	S1710307099	2018-11-13 22:05:27	S1710307099	2018-11-13 22:05:27

Illustration 6: p_num = 3

3. Performance-Optimierung

```
DECLARE
  starttime NUMBER;
  total NUMBER;
BEGIN
  starttime := DBMS_UTILITY.GET_TIME();
  UPDATE my_payment mp
  SET penalty = amount * 1.15
  WHERE EXISTS (SELECT r.rental_id
          FROM rental r
                 INNER JOIN inventory i ON r.inventory_id = i.inventory_id
                 INNER JOIN film f ON i.film_id = f.film_id
          WHERE r.rental_id = mp.rental_id AND
            CEIL(return_date - rental_date) > f.rental_duration);
  total := DBMS_UTILITY.GET_TIME() - starttime;
  DBMS_OUTPUT.PUT_LINE('PL/SQL WITHOUT LOOP: ' || total / 100 || ' seconds');
END;
/
```

```
[2018-11-13 22:08:05] completed in 280 ms
[2018-11-13 22:08:05] PL/SQL WITHOUT LOOP: .26 seconds
```

Illustration 7: Optimierte Laufzeit

4. Multiple Choice

1. PL/SQL eignet sich gut um...

B: SQL-Anweisungen in Verbindung mit Schleifen und Bedingungen auszuführen.

C: wiederkehrende Aufgaben auszuführen.

2. In PL/SQL...

C: sind SQL-Funktionen (zB Datum) ebenfalls verfügbar.

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

B: sind spezielle Schlüsselwörter (INTO, …) für die Speicherung eines Ergebnisses notwendig

D: können diese mit anderen PL/SQL-Konstrukten gemischt werden.

4. Welche Aussagen sind wahr?

A: Liefert ein SQL-Statement mehrere Ergebniszeilen, ist ein Cursor notwendig.

B: Eine Variable kann auch als „NOT NULL“ deklariert werden.

E: Mit PL/SQL soll möglichst viele Business-Logik in die Datenbank gebracht werden. (Stimmt großteils; Zentralisierung der Logik hilft um Applikationen auf verschiedenen Platformen eine gemeinsame Schnittstelle zu bieten.)

F: PL/SQL kann auch Java-Code ausführen.

G: IN- und OUT-Parameter einer Prozedur können einen Default-Wert besitzen.