

Abgabetermin: 21.11.2018, 13:30 Uhr

<input checked="" type="checkbox"/> DES31UE Niklas	Name <u>Papesh Konstantin</u>	Aufwand in h <u>8</u>
<input type="checkbox"/> DES32UE Niklas		
<input type="checkbox"/> DES32UE Traxler	Punkte _____	Kurzzeichen Tutor _____

Ziel dieser Übung ist die Vertiefung von PL/SQL in Paketen, dem Cursor-Konzept und Behandlung von Exceptions.

**1. Datenbankpakete (PL/SQL-Packages)****(10 Punkte – 4+6 Pkt)**

Mit Hilfe von PL/SQL-Packages können Sie zusammengehörige PL/SQL-Typen, Variablen, Datenstrukturen, Exceptions und Unterprogramme in einer Bibliothek zusammenfassen. Packages bestehen normalerweise aus zwei Komponenten (Spezifikation und Body), die separat in der Datenbank gespeichert werden. Das Package selbst kann nicht aufgerufen, parametrisiert oder verschachtelt werden.

```
CREATE OR REPLACE PACKAGE top_customer_pkg AS
```

```
  ...  
END;  
/
```

```
CREATE OR REPLACE PACKAGE BODY top_customer_pkg AS
```

```
  ...  
END;  
/
```

1. Erstellen Sie ein Datenbankpaket `top_customer_pkg` mit der Funktion `GetFilmcount` (übergeben Sie eine Customer-Id als Parameter) und speichern Sie die Package Spezifikation und den Package Body gemeinsam ab. Erstellen Sie einen Testaufruf (zB für `customer_id = 100`). `GetFilmcount` soll aus der Sakila Datenbank die Anzahl der geliehenen Filme ermitteln, dazu werden nur Filme gezählt, die mindestens 60 Minuten lang sind.
2. Implementieren Sie nun im Package `top_customer_pkg` eine zusätzliche Prozedur `GetTopNCustomers`, die drei Eingabeparameter verlangt:

- die Anzahl der Kunden `n_count`, um eine Top-N Liste der `n_count` besten Kunden ausgeben zu können
- `begin_date` und
- `end_date`, um die Auswahl von Datensätzen auf den Bereich `begin_date <= rental_date <= end_date` einschränken zu können.

D.h. es sollen die Anzahl (`n_count`) Kunden ausgegeben werden, die die meisten Filme zwischen `begin_date` und `end_date` ausgeliehen haben. Wird keine Anzahl an Kunden angegeben, so sollen default-mäßig die Top-10-Kunden ausgegeben werden. Geben Sie auch für die anderen Parameter Default-Werte an, um einen Aufruf ohne Parameter zu ermöglichen.

Verwenden Sie für die zu erstellende Prozedur eine `CURSOR FOR LOOP`, die Prozedur `DBMS_OUTPUT.PUT_LINE` und erstellen Sie zur Ermittlung der Film-Anzahl eine Erweiterung der bereits vorhandenen Funktion (verwenden Sie dazu Overloading/Überladen mit `begin_date`

und end\_date). Sortieren Sie nach der Anzahl absteigend. Geben Sie die Package Spezifikation, den Package Rumpf sowie den Testaufruf (mit und ohne Parameter) an.

#### Beispielausgabe

```
...  
The top 10 customers from 01.01.07 to 31.12.16 are:  
ELEANOR HUNT: 44 films  
...
```

## 2. Cursor mit FOR-UPDATE

(10 Punkte – 1.-2. je 3 Pkt und 3.-4. je 2 Pkt)

Wenn mehrere Sessions für eine einzelne Datenbank vorhanden sind, besteht die Möglichkeit, dass die Zeilen einer bestimmten Tabelle aktualisiert wurden, nachdem Sie den Cursor geöffnet haben. Sie sehen die aktualisierten Daten nur, wenn Sie den Cursor erneut öffnen. Es ist daher günstiger, die Zeilen zu sperren, bevor Sie Zeilen aktualisieren oder löschen. Sie können zum Sperren der Zeilen die FOR-UPDATE-Klausel in der Cursor-Abfrage verwenden.

1. Erstellen Sie eine Tabelle top\_customers, die customer\_id und die Anzahl der Filme enthält. Speichern Sie in der Tabelle auch den User und das Datum an dem der jeweilige Datensatz eingefügt wurde. Eine weitere Spalte enthält das Datum an dem der Datensatz ‚deaktiviert‘ wurde (dh. die Person nicht mehr zu den Top Customers gehört), für aktive Datensätze ist dieser Eintrag NULL.  
Erweitern Sie Ihre Prozedur GetTopNCustomers dahingehend, dass zusätzlich zur Ausgabe mit DMBS\_OUTPUT auch Datensätze in die Tabelle top\_customers eingefügt werden.
2. Fügen Sie dem Datenbankpaket top\_customer\_pkg eine weitere Prozedur DeactivateTopCustomers hinzu, die Datensätze in der Tabelle top\_customers mit weniger als einer angegebenen Anzahl an Filmen deaktiviert (das aktuelle Datum einträgt). Testen Sie Ihre Prozedur.

Hinweise: Verwenden Sie beim Cursor die FOR-UPDATE und beim UPDATE die WHERE CURRENT OF-Klausel. Geben Sie die Package Spezifikation, den Package Body sowie den Testaufruf an.

3. Öffnen Sie eine zweite Datenbank-Session (Strg+Shift+N) und führen Sie die Prozedur DeactivateTopCustomers mit den gleichen Parametern in jeder Session aus, ohne ein COMMIT auszuführen. Wählen Sie die Parameter so, damit zumindest ein Satz aus top\_customers selektiert wird. Was passiert in der zweiten Session? Führen Sie in der ersten ein COMMIT aus und beschreiben Sie die Auswirkungen.
4. Erweitern Sie nun den Cursor in der Prozedur DeactivateTopCustomers um die NOWAIT-Klausel. Wiederholen Sie den Test und erläutern Sie den Unterschied. Geben Sie die Package Spezifikation, den Package Rumpf sowie den Testaufruf an.

### 3. EXCEPTIONS

(4 Punkte – 1. 2 Pkt und 2.-3. je 1 Pkt)

Diese Aufgabe behandelt die Verwendung von vordefinierten Exceptions.

1. Erstellen Sie eine PL/SQL-Prozedur mit dem Parameter `name_part` (vom Typ `customer.last_name`), die anhand eines angegebenen Namens(-teils) jene Kunden auswählt, deren Namen damit beginnen. Hinweis: Verwenden Sie keinen expliziten Cursor und vergleichen Sie mit `LIKE`. Wenn der an die Prozedur übergebene Namensteil nur eine Zeile zurückgibt, fügen Sie in die Tabelle `messages` den Namen des Kunden (Vor- und Nachname und `customer_id`) ein. Die Tabelle `messages` soll aus einer Spalte `results` vom Typ `VARCHAR2(100)` bestehen. Testen Sie die Prozedur anhand folgender Fälle: a) Keine Zeile wird selektiert. b) Eine Zeile wird selektiert. c) Mehrere Zeilen werden selektiert. Was stellen Sie fest?
2. Wenn der eingegebene Name keine Zeilen zurückgibt, behandeln Sie die Exception mit einem entsprechenden Exception Handler, und fügen Sie in die Tabelle `messages` die Meldung „No customer found where last name begins with <name>“ ein.
3. Behandeln Sie beliebige weitere Exceptions mit einem entsprechenden Exception Handler, und fügen Sie in die Tabelle `messages` die Meldung „An undefined error occurred“ ein.

## 1.1

```
CREATE OR REPLACE PACKAGE top_customer_pkg AS
  FUNCTION GetFilmcount(cus_id customer.customer_id%TYPE)
    RETURN INTEGER;
END;
```

```
CREATE OR REPLACE PACKAGE BODY top_customer_pkg AS
  FUNCTION GetFilmcount(cus_id customer.customer_id%TYPE)
    RETURN INTEGER
  IS
    film_count INTEGER;
  BEGIN
    SELECT COUNT(*) INTO film_count
    FROM customer
      INNER JOIN rental r2 ON customer.customer_id = r2.customer_id
      INNER JOIN inventory i2 ON r2.inventory_id = i2.inventory_id
      INNER JOIN film f ON i2.film_id = f.film_id
    WHERE customer.customer_id = cus_id
      AND length > 60;
    RETURN film_count;
  END;
END;
```

```
BEGIN
  dbms_output.PUT_LINE('Checking film count for ID 100');
  dbms_output.PUT_LINE(top_customer_pkg.GetFilmcount(100));
END;
```

```
[2018-11-20 14:09:46] Checking film count for ID 100
[2018-11-20 14:09:46] 22
```

## 1.2

```
CREATE OR REPLACE PACKAGE top_customer_pkg AS
  FUNCTION GetFilmcount(cus_id customer.customer_id%TYPE, begin_date rental.rental_date%TYPE,
    end_date rental.rental_date%TYPE)
    RETURN INTEGER;
  FUNCTION GetFilmcount(cus_id customer.customer_id%TYPE)
    RETURN INTEGER;
  PROCEDURE GetTopNCustomer(n_count INTEGER DEFAULT 10, begin_date rental.rental_date%TYPE DEFAULT
    SYSDATE-7, end_date rental.rental_date%TYPE DEFAULT SYSDATE);
END;
```

```
CREATE OR REPLACE PACKAGE BODY top_customer_pkg AS
  FUNCTION GetFilmcount(cus_id customer.customer_id%TYPE, begin_date rental.rental_date%TYPE,
    end_date rental.rental_date%TYPE)
    RETURN INTEGER
  IS
    film_count INTEGER;
  BEGIN
    SELECT COUNT(*) INTO film_count
    FROM customer
      INNER JOIN rental r2 ON customer.customer_id = r2.customer_id
```

```

        INNER JOIN inventory i2 ON r2.inventory_id = i2.inventory_id
        INNER JOIN film f ON i2.film_id = f.film_id
    WHERE customer.customer_id = cus_id
        AND length > 60 AND begin_date <= rental_date AND rental_date <= end_date;
    RETURN film_count;
END;

FUNCTION GetFilmcount(cus_id customer.customer_id%TYPE)
    RETURN INTEGER
IS
    BEGIN
        RETURN GetFilmcount(cus_id, TO_DATE('1900-01-01', 'yyyy-mm-dd'), SYSDATE);
    END;

PROCEDURE GetTopNCustomer(n_count INTEGER DEFAULT 10, begin_date rental.rental_date%TYPE DEFAULT
SYSDATE-7, end_date rental.rental_date%TYPE DEFAULT SYSDATE)
IS
    film_count INTEGER;
    BEGIN
        DBMS_OUTPUT.PUT_LINE('The top ' || n_count || ' customers from ' || begin_date || ' to ' ||
end_date || ' are:');
        FOR cus_record IN (
            SELECT first_name, last_name, c.customer_id
            FROM customer c
                INNER JOIN rental r2 ON c.customer_id = r2.customer_id
                INNER JOIN inventory i2 ON r2.inventory_id = i2.inventory_id
                INNER JOIN film f ON i2.film_id = f.film_id
            WHERE length > 60
                AND begin_date <= rental_date AND rental_date <= end_date
            GROUP BY first_name, last_name, c.customer_id
            ORDER BY COUNT(f.film_id) DESC
            FETCH FIRST n_count ROWS ONLY
        )
        LOOP
            film_count := GetFilmcount(cus_record.customer_id, begin_date, end_date);
            DBMS_OUTPUT.PUT_LINE(cus_record.first_name || ' ' || cus_record.last_name || ': ' ||
film_count || ' films');
            INSERT INTO top_customers(cus_id, count, created_by) VALUES (cus_record.customer_id,
film_count, USER);
        END LOOP;
    END;
END;

BEGIN
    dbms_output.PUT_LINE(top_customer_pkg.GetFilmcount(100));
    top_customer_pkg.GetTopNCustomer(20, TO_DATE('2007-01-01', 'yyyy-mm-dd'),
TO_DATE('2016-01-31', 'yyyy-mm-dd'));
    top_customer_pkg.GetTopNCustomer();
END;

```

```

[2018-11-20 14:13:25] The top 20 customers from 01-JAN-07 to 31-JAN-16 are:
[2018-11-20 14:13:25] ELEANOR HUNT: 44 films
[2018-11-20 14:13:25] KARL SEAL: 41 films
[2018-11-20 14:13:25] MARCIA DEAN: 37 films
[2018-11-20 14:13:25] RHONDA KENNEDY: 37 films
[2018-11-20 14:13:25] WESLEY BULL: 37 films
[2018-11-20 14:13:25] CLARA SHAW: 37 films

```

```
[2018-11-20 14:13:25] TAMMY SANDERS: 36 films
[2018-11-20 14:13:25] MARION SNYDER: 36 films
[2018-11-20 14:13:25] SUE PETERS: 35 films
[2018-11-20 14:13:25] DAISY BATES: 35 films
[2018-11-20 14:13:25] MARSHA DOUGLAS: 34 films
[2018-11-20 14:13:25] CURTIS IRBY: 34 films
[2018-11-20 14:13:25] ELIZABETH BROWN: 34 films
[2018-11-20 14:13:25] TOMMY COLLAZO: 34 films
[2018-11-20 14:13:25] JESSICA HALL: 34 films
[2018-11-20 14:13:25] ANGELA HERNANDEZ: 33 films
[2018-11-20 14:13:25] BRANDON HUEY: 33 films
[2018-11-20 14:13:25] MIKE WAY: 33 films
[2018-11-20 14:13:25] JUSTIN NGO: 33 films
[2018-11-20 14:13:25] MICHELLE CLARK: 33 films
[2018-11-20 14:13:25] The top 10 customers from 13-NOV-18 to 20-NOV-18 are:
```

## 2.1

```
DROP TABLE top_customers;
CREATE TABLE top_customers(cus_id NUMBER NOT NULL, count INTEGER NOT NULL, created_by VARCHAR(30)
NOT NULL, date_added TIMESTAMP DEFAULT SYSDATE NOT NULL, date_removed TIMESTAMP DEFAULT NULL);

BEGIN
  dbms_output.PUT_LINE(top_customer_pkg.GetFilmcount(100));
  top_customer_pkg.GetTopNCustomer(20, TO_DATE('2007-01-01','yyyy-mm-dd'),
TO_DATE('2016-01-31','yyyy-mm-dd'));
  top_customer_pkg.GetTopNCustomer();
END;

SELECT * FROM top_customers;
```

148	44	S1720307111	2018-11-20 14:11:40.000000	<null>
526	41	S1720307111	2018-11-20 14:11:40.000000	<null>
236	37	S1720307111	2018-11-20 14:11:40.000000	<null>
137	37	S1720307111	2018-11-20 14:11:40.000000	<null>
469	37	S1720307111	2018-11-20 14:11:40.000000	<null>
144	37	S1720307111	2018-11-20 14:11:40.000000	<null>
75	36	S1720307111	2018-11-20 14:11:40.000000	<null>
178	36	S1720307111	2018-11-20 14:11:40.000000	<null>
197	35	S1720307111	2018-11-20 14:11:40.000000	<null>
295	35	S1720307111	2018-11-20 14:11:40.000000	<null>

40 Zeilen

## 2.2

```
CREATE OR REPLACE PACKAGE top_customer_pkg AS
  FUNCTION GetFilmcount(cus_id customer.customer_id%TYPE, begin_date rental.rental_date%TYPE,
end_date rental.rental_date%TYPE)
    RETURN INTEGER;
  FUNCTION GetFilmcount(cus_id customer.customer_id%TYPE)
    RETURN INTEGER;
  PROCEDURE GetTopNCustomer(n_count INTEGER DEFAULT 10, begin_date rental.rental_date%TYPE DEFAULT
SYSDATE-7, end_date rental.rental_date%TYPE DEFAULT SYSDATE);
  PROCEDURE DeactivateTopCustomers(n_count INTEGER);
```

END;

```
CREATE OR REPLACE PACKAGE BODY top_customer_pkg AS
  FUNCTION GetFilmcount(cus_id customer.customer_id%TYPE, begin_date rental.rental_date%TYPE,
end_date rental.rental_date%TYPE)
```

RETURN INTEGER

IS

film\_count INTEGER;

BEGIN

SELECT COUNT(\*) INTO film\_count

FROM customer

INNER JOIN rental r2 ON customer.customer\_id = r2.customer\_id

INNER JOIN inventory i2 ON r2.inventory\_id = i2.inventory\_id

INNER JOIN film f ON i2.film\_id = f.film\_id

WHERE customer.customer\_id = cus\_id

AND length > 60 AND begin\_date <= rental\_date AND rental\_date <= end\_date;

RETURN film\_count;

END;

```
FUNCTION GetFilmcount(cus_id customer.customer_id%TYPE)
```

RETURN INTEGER

IS

BEGIN

RETURN GetFilmcount(cus\_id, TO\_DATE('1900-01-01', 'yyyy-mm-dd'), SYSDATE);

END;

```
PROCEDURE GetTopNCustomer(n_count INTEGER DEFAULT 10, begin_date rental.rental_date%TYPE DEFAULT
SYSDATE-7, end_date rental.rental_date%TYPE DEFAULT SYSDATE)
```

IS

film\_count INTEGER;

BEGIN

```
DBMS_OUTPUT.PUT_LINE('The top ' || n_count || ' customers from ' || begin_date || ' to ' ||
end_date || ' are:');
```

FOR cus\_record IN (

SELECT first\_name, last\_name, c.customer\_id

FROM customer c

INNER JOIN rental r2 ON c.customer\_id = r2.customer\_id

INNER JOIN inventory i2 ON r2.inventory\_id = i2.inventory\_id

INNER JOIN film f ON i2.film\_id = f.film\_id

WHERE length > 60

AND begin\_date <= rental\_date AND rental\_date <= end\_date

GROUP BY first\_name, last\_name, c.customer\_id

ORDER BY COUNT(f.film\_id) DESC

FETCH FIRST n\_count ROWS ONLY

)

LOOP

film\_count := GetFilmcount(cus\_record.customer\_id, begin\_date, end\_date);

```
DBMS_OUTPUT.PUT_LINE(cus_record.first_name || ' ' || cus_record.last_name || ': ' ||
film_count || ' films');
```

```
INSERT INTO top_customers(cus_id, count, created_by) VALUES (cus_record.customer_id,
film_count, USER);
```

END LOOP;

END;

```
PROCEDURE DeactivateTopCustomers(n_count INTEGER)
```

IS

CURSOR cus\_cur (n\_count INTEGER) IS

```

SELECT date_removed FROM top_customers WHERE count < n_count
FOR UPDATE NOWAIT;
result NUMBER;
BEGIN
  SELECT count(*) INTO result
  FROM top_customers WHERE count < n_count;
  IF(result > 0) THEN
    FOR the_cus IN cus_cur(n_count)
      LOOP
        UPDATE top_customers SET date_removed = SYSDATE
        WHERE CURRENT OF cus_cur;
      END LOOP;
    END IF;
  END;
END;

BEGIN
  top_customer_pkg.DeactivateTopCustomers(35);
END;
SELECT * FROM top_customers;

```

178	36	S1720307111	2018-11-20 14:11:40.000000	<null>
197	35	S1720307111	2018-11-20 14:11:40.000000	<null>
295	35	S1720307111	2018-11-20 14:11:40.000000	<null>
257	34	S1720307111	2018-11-20 14:11:40.000000	2018-11-20 14:15:26...
410	34	S1720307111	2018-11-20 14:11:40.000000	2018-11-20 14:15:26...
5	34	S1720307111	2018-11-20 14:11:40.000000	2018-11-20 14:15:26...

## 2.3

Wird der Befehl auf einer Konsole ausgeführt, wird die Tabelle in der anderen Session zurückgesetzt.

Wird in der ersten ein COMMIT ausgeführt, übernimmt die zweite Session die Änderungen der ersten.

## 2.4

Folgende Fehlermeldung tritt auf: ORA-00054: resource busy and acquire with NOWAIT specified or timeout expired

Die Tabelle ist nun so lange gesperrt, bis in der verändernden Konsole ein COMMIT Befehl ausgeführt wird.

## 3.1

```

DROP TABLE messages;
CREATE TABLE messages(results VARCHAR2(100));

CREATE OR REPLACE PROCEDURE name_part(name customer.last_name%TYPE)
IS
  first customer.first_name%TYPE;
  last customer.last_name%TYPE;

```



```

cus_id customer.customer_id%TYPE;
BEGIN
  SELECT first_name, last_name, customer_id INTO first, last, cus_id FROM customer WHERE
last_name LIKE name;
  IF SQL%ROWCOUNT = 1 THEN
    INSERT INTO messages(results) VALUES (first || ' ' || last || ' ' || cus_id);
  END IF;
  DBMS_OUTPUT.PUT_LINE('ROWCOUNT ' || SQL%ROWCOUNT);
END;

CALL name_part('FFFF'); -- No name found
CALL name_part('FULTZ'); -- Einer
CALL NAME_PART('ALL%'); -- Mehrere
SELECT * FROM messages;

-- Keiner: Fehlermeldung 'no data found'
-- Einer: CALL wird ausgeführt, Nachricht wird eingetragen
-- Mehrere: Fehlermeldung 'exact fetch returns more than requested number of rows'

```

```
GERALD FULTZ 356
```

## 3.2

```

CREATE OR REPLACE PROCEDURE name_part(name customer.last_name%TYPE)
IS
  first customer.first_name%TYPE;
  last customer.last_name%TYPE;
  cus_id customer.customer_id%TYPE;
BEGIN
  SELECT first_name, last_name, customer_id INTO first, last, cus_id FROM customer WHERE
last_name LIKE name;
  INSERT INTO messages(results) VALUES (first || ' ' || last || ' ' || cus_id);
  EXCEPTION
  WHEN NO_DATA_FOUND THEN
    INSERT INTO messages(results) VALUES ('No customer found where last name begins with ' ||
name);
  END;

```

```
No customer found where last name begins with FFFF
```

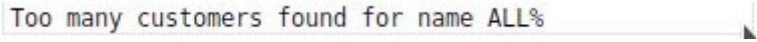
## 3.3

```

CREATE OR REPLACE PROCEDURE name_part(name customer.last_name%TYPE)
IS
  first customer.first_name%TYPE;
  last customer.last_name%TYPE;
  cus_id customer.customer_id%TYPE;
BEGIN
  SELECT first_name, last_name, customer_id INTO first, last, cus_id FROM customer WHERE
last_name LIKE name;
  INSERT INTO messages(results) VALUES (first || ' ' || last || ' ' || cus_id);
  EXCEPTION
  WHEN NO_DATA_FOUND THEN

```

```
INSERT INTO messages(results) VALUES ('No customer found where last name begins with ' ||  
name);  
WHEN TOO_MANY_ROWS THEN  
INSERT INTO messages(results) VALUES ('Too many customers found for name ' || name);  
WHEN OTHERS THEN  
INSERT INTO messages(results) VALUES ('An undefined error occurred');  
DBMS_OUTPUT.PUT_LINE(SQLCODE || SQLERRM);  
END;
```

A screenshot of a database query result. It shows a single line of text: "Too many customers found for name ALL%". The text is displayed in a light blue font on a white background, with a small cursor icon at the end of the line.