

Abgabetermin: 17.11.2021

<input type="radio"/>	DES3UEG1: Niklas	Name	Angelos Angelis	Aufwand in h	5
<input checked="" type="radio"/>	DES3UEG2: Niklas				
<input type="radio"/>	DES3UEG3: 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. Zudem sollen Unterschiede zwischen SQL und PL/SQL erkannt werden.

Zusätzliche Hinweise

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

1. PL/SQL Grundlagen (6 Punkte – 0,5 + 1 + 1 + 1,5 + 2)

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 dazu die Tabelle `top_salaries` um das Feld `emp_cnt` und wählen Sie einen passenden Datentyp aus. Definieren Sie einen künstlichen Primärschlüssel als generierte Identität und ein Check Constraint zur Sicherstellung, dass `emp_cnt` größer als Null ist. Verwenden Sie für die Erweiterung ALTER-Befehle (leeren Sie Ihre Tabelle davor).

Hinweis: Verwenden Sie das DataDictionary um Constraints anzuzeigen.

```
SELECT *  
FROM user_cons_columns  
      INNER JOIN user_constraints c USING (constraint_name)  
WHERE c.table_name = 'TOP_SALARIES';
```

5. Modifizieren Sie das Skript `UE06_01_02.sql`, um das neue Feld korrekt zu befüllen.

2. PL/SQL Prozeduren

(6 Punkte – 1 + 3 + 2)

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.
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. 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.

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.

Hinweis: Nutzen Sie SET serveroutput ON oder ein DBMS-Output-Panel um die gemessenen Zeiten zu sehen.

```
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. Regelmäßige Aufgaben (Teil 2)

(3 Punkte – 2 + 1)

Sie erhalten die Aufgabe, die Aktivität in der Datenbank zu analysieren. Als ersten Anlaufpunkt möchten Sie die Anzahl der offenen Benutzer-Sessions überwachen.

1. Erstellen Sie einen Job mit dem Oracle-Scheduler, der Ihr Skript aus Bsp. 5.3 der Übung 5 regelmäßig ausführt und so die Anzahl der offenen Sessions protokolliert. Rufen Sie Ihr Statement im Rahmen eines PL/SQL-Blocks alle 30 Minuten auf und stellen Sie gleichzeitig sicher, dass der Job nach sieben Tagen nicht weiter ausgeführt wird (end_date).
2. Erstellen Sie eine analytische Abfrage, die auf Ihre Tabelle monitor_sessions einen gleitenden Durchschnitt (innerhalb 3 Stunden zuvor und 3 Stunden danach) ermittelt.

5. Multiple Choice

(4 Punkte)

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 den gleichen Namen besitzen wie Tabellen oder Spalten.
- ☒ kann von der Tabelle dual nicht selektiert werden.
- ☒ sind SQL-Funktionen (zB Datum) ebenfalls verfügbar.
- ☒ führt COMMIT zu einer Fehlermeldung.

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 nicht mehrere 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.

Aufgabe 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));
```

```
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;
```

SQL

PL/SQL

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.

Antwort: wenn man $n=0$ eingibt wird garnichts in der Tabelle employees eingetragen.

Aufgrund des Cursors werden maximal 18 Tupel eingefügt. Wenn man mehr als 18 angibt werden trotzdem nur 18 eingefügt

```
SELECT * FROM top_salaries;
-- Zusätzlich zum Gehalt soll auch die Anzahl der Mitarbeiter abgespeichert
-- werden, die dieses
-- Gehalt verdienen. Erweitern Sie dazu die Tabelle top_salaries um das Feld
emp_cnt und wählen
-- Sie einen passenden Datentyp aus. Definieren Sie einen künstlichen
Primärschlüssel als
-- generierte Identität und ein Check Constraint zur Sicherstellung, dass
emp_cnt größer als Null ist.
-- Verwenden Sie für die Erweiterung ALTER-Befehle (leeren Sie Ihre Tabelle
davor).
DROP TABLE top_salaries;
```

	SALARY	ID	EMP_COUNT	CREATEDBY	DATECREATED	MODIFIEDBY	DATEMODIFIED
1	24000.00	12	1	S2010307048	2021-11-11 21:48:35.000000	S2010307048	2021-11-11 21:48:35.000000
2	17000.00	13	2	S2010307048	2021-11-11 21:48:35.000000	S2010307048	2021-11-11 21:48:35.000000
3	13000.00	14	1	S2010307048	2021-11-11 21:48:35.000000	S2010307048	2021-11-11 21:48:35.000000
4	12000.00	15	1	S2010307048	2021-11-11 21:48:35.000000	S2010307048	2021-11-11 21:48:35.000000
5	11000.00	16	1	S2010307048	2021-11-11 21:48:35.000000	S2010307048	2021-11-11 21:48:35.000000
6	10500.00	17	1	S2010307048	2021-11-11 21:48:35.000000	S2010307048	2021-11-11 21:48:35.000000

```

ALTER TABLE top_salaries ADD
(
  id INT GENERATED BY DEFAULT AS IDENTITY
    ( START WITH 1 INCREMENT BY 1
      MINVALUE 1 MAXVALUE 2000) PRIMARY KEY ,
  emp_count NUMBER(5) CHECK (emp_count > 0)
);

-- Modifizieren Sie das Skript UE06_01_02.sql, um das neue Feld korrekt zu
-- befüllen.
DELETE FROM top_salaries;
DECLARE
num NUMBER(3) := &p_num;
CURSOR emp_cursor IS
  SELECT salary, count(*) as emp_cnt
  FROM employees
  GROUP BY salary
  ORDER BY salary DESC;
sal top_salaries.salary%TYPE;
emp_cnt top_salaries.emp_count%TYPE;
BEGIN
OPEN emp_cursor;
FETCH emp_cursor INTO sal, emp_cnt;
WHILE emp_cursor%ROWCOUNT <= num AND emp_cursor%FOUND LOOP
INSERT INTO top_salaries
VALUES (sal,DEFAULT, emp_cnt);
FETCH emp_cursor INTO sal, emp_cnt;
END LOOP;
CLOSE emp_cursor;
END;
Aufgabe2)

-- 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 Modified-Felder ident.
DELETE FROM top_salaries;
ALTER TABLE top_salaries ADD
(
  createdBy VARCHAR2(30),
  dateCreated TIMESTAMP,
  modifiedBy VARCHAR2(30),
  dateModified TIMESTAMP
);
-- 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. Die
-- Prozedur soll folgende Spezifikation aufweisen:
CREATE OR REPLACE PROCEDURE InsertTopSalaries(pSalary IN NUMBER,pEmp_cnt IN

```

```

NUMBER)
IS
    cBy TOP_SALARIES.createdBy%type := USER;
--    modBy TOP_SALARIES.modifiedBy%type := USER;
    daCr TOP_SALARIES.dateCreated%type := SYSDATE;
--    daMod TOP_SALARIES.dateModified%type := SYSDATE;
BEGIN
    INSERT INTO TOP_SALARIES (SALARY, ID, EMP_COUNT, CREATEDBY, DATECREATED,
MODIFIEDBY, DATEMODIFIED)
        VALUES (pSalary, DEFAULT, pEmp_cnt, cby, daCr, cBy, daCr);
end;

CALL InsertTopSalaries(2000,1);

SELECT * FROM TOP_SALARIES;
-- Ersetzen Sie die INSERT-Anweisung im Skript UE06_01_02 durch die in der
vorherigen
-- Aufgabe erstellten Prozedur InsertTopSalaries und überprüfen Sie das
Ergebnis.
-- Hinweis: Um auch die Uhrzeit zu sehen, können Sie das Datumsformat mit
folgendem
-- Kommando festlegen.
DELETE FROM top_salaries;
DECLARE
    num NUMBER(3) := &p_num;
    CURSOR emp_cursor IS
        SELECT salary, count(*) as emp_cnt
        FROM employees
        GROUP BY salary
        ORDER BY salary DESC;
    sal top_salaries.salary%TYPE;
    emp_cnt top_salaries.emp_count%TYPE;
BEGIN
    OPEN emp_cursor;
    FETCH emp_cursor INTO sal, emp_cnt;
    WHILE emp_cursor%ROWCOUNT <= num AND emp_cursor%FOUND LOOP
        InsertTopSalaries(sal, emp_cnt);
        FETCH emp_cursor INTO sal, emp_cnt;
    END LOOP;
    CLOSE emp_cursor;
END;

```

Aufgabe3)

```

-- 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.
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;

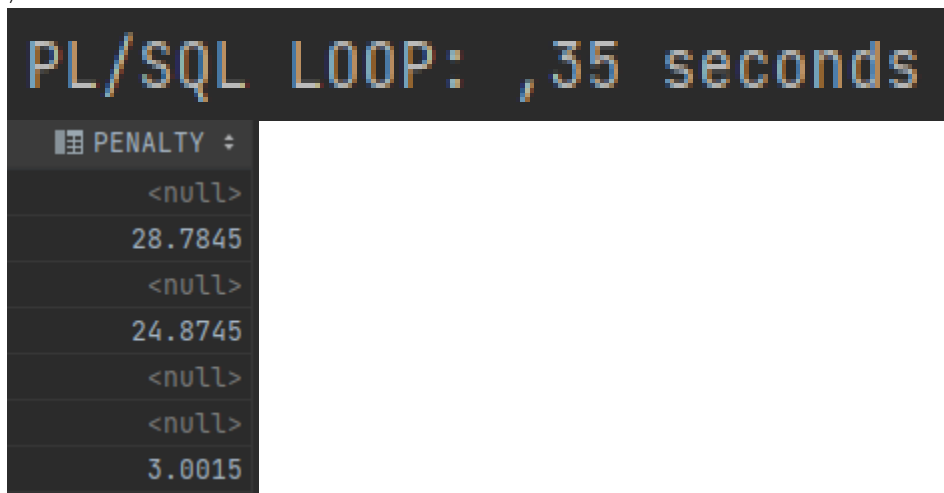
-- UPDATE in loop
DECLARE
    starttime NUMBER;
    total NUMBER;
    maxRent NUMBER := 0;
    actualRent NUMBER := 0;
BEGIN
    starttime := DBMS_UTILITY.GET_TIME();
    FOR mp IN (SELECT amount, rental_id, payment_id, payment_date FROM
my_payment) LOOP
        SELECT MAX(rental_duration) INTO maxRent
        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;
/

```



PENALTY
<null>
28.7845
<null>
24.8745
<null>
<null>
3.0015

```
DROP TABLE my_payment;
```

```
-- 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.

```
SELECT * FROM my_payment;
```

```
DECLARE
```

```
    starttime NUMBER;
```

```
    total NUMBER;
```

```
    maxRent NUMBER := 0;
```

```
    actualRent NUMBER := 0;
```

```
BEGIN
```

```
    starttime := DBMS_UTILITY.GET_TIME();
```

```
    UPDATE my_payment
```

```
    SET penalty = amount * 1.15
```

```
    WHERE (SELECT MAX(rental_duration)
```

```
        FROM film
```

```
        INNER JOIN inventory USING (film_id)
```

```
        INNER JOIN rental USING (inventory_id) WHERE RENTAL_ID =
```

```
my_payment.RENTAL_ID)
```

```
    <
```

```
    (SELECT MAX(CEIL(return_date - rental_date))
```

```
        FROM rental
```

```
        WHERE RENTAL_ID = my_payment.RENTAL_ID);
```

```
    total := DBMS_UTILITY.GET_TIME() - starttime;
```

```
    DBMS_OUTPUT.PUT_LINE('PL/SQL LOOP: ' || total / 100 || ' seconds');
```

```
END;
```

```
/
```

```
ROLLBACK ;
```

WITHOUT LOOP: ,1 seconds

PENALTY
<null>
28.7845
<null>
24.8745
<null>
<null>
3.0015

AUFGABE4)

```
INSERT INTO monitor_sessions(MS_ID,COUNT, LAST_CHECK)
```

```
VALUES (
```

```
DEFAULT,
```

```
(SELECT COUNT(SID)
```

```
FROM V$SESSION
```

```
WHERE SCHEMANAME <> 'SYS'),
```

```
SYSDATE
```

```
);
```

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

BEGIN

```

DBMS_SCHEDULER.CREATE_SCHEDULE(
    schedule_name => 'weekcycle'
    , start_date => '14/11/2021 09:30:00' -- frühester Start
    , repeat_interval => 'FREQ=MINUTELY; INTERVAL=30;' -- jeden Tag
    um 15, 16, 17 und 18 Uhr
    , end_date => SYSDATE + 7
    , comments => 'every 30 minutes');
END;

```

BEGIN

```

DBMS_SCHEDULER.CREATE_JOB(
    job_name => 'Countsessionsweekly'
    , job_type => 'PLSQL_BLOCK'
    , job_action => '
        DECLARE INSERT INTO
monitor_sessions(MS_ID,COUNT, LAST_CHECK)
        VALUES (
        DEFAULT,
        (SELECT COUNT(SID)
        FROM V$SESSION
        WHERE USERNAME IS NOT NULL,
        SYSDATE
        );' -- Name der Prozedur
    , schedule_name => 'weekcycle' -- Schedule verwenden
    , enabled => TRUE
    , comments => 'Insert Session count every 30 Minutes');
END;

```

```

SELECT *
FROM monitor_sessions;

```

```

SELECT MS_ID, LAST_CHECK, COUNT, AVG(COUNT) OVER (ORDER BY LAST_CHECK ROWS
BETWEEN 6 PRECEDING AND 6 FOLLOWING) AS AVG
FROM MONITOR_SESSIONS;

```

	MS_ID	LAST_CHECK	COUNT	AVG
1	260	2021-11-14 09:02:46	8	22.57142857142857142857142857142857
2	261	2021-11-14 18:35:50	19	23.125
3	262	2021-11-14 18:52:25	23	23.125
4	280	2021-11-17 11:51:07	27	23.125
5	281	2021-11-17 11:51:09	27	23.125
6	282	2021-11-17 11:51:10	27	23.125
7	283	2021-11-17 11:51:11	27	23.125
8	284	2021-11-17 11:51:12	27	25.28571428571428571428571428571429