

<input checked="" type="checkbox"/> DES3UEG1: Glock	Name <u>Elias Leonhardsberger</u>	Aufwand in h <u>4</u>
<input type="checkbox"/> DES3UEG2: Werth	Punkte _____	Kurzzeichen Tutor _____

Ziel dieser Übung ist die Vertiefung des Trigger-Konzepts und die praktische Anwendung der unterschiedlichen Typen.

1. Trigger (Sakila)**(4+3+2+3 = 12 Punkte)**

Die Zeilen in der Tabelle PAYMENT speichern den Bezahlvorgang nachdem ein Film zurückgegeben wurde. Erstellen Sie in PL/SQL ein Programm, das sicherstellt, dass bei INSERT- und UPDATE-Operationen auf diese Tabelle der bezahlte Betrag nicht die vorgesehene Rate eines Films überschreitet. Beachten Sie NULL-Werte!

1. Erstellen Sie eine Prozedur mit dem Namen CHECK_AMOUNT mit folgenden Parametern:

- die Film-Id
- den bezahlten Betrag (wird eingelesen **und** ggf. geändert zurückgegeben)
- die Anzahl der verliehenen Tage

Verwenden Sie die Film-Id, um die Verleihrate (rental_rate) für den angegebenen Film zu finden. Vergleichen Sie den bezahlten Betrag mit dem tatsächlich zu bezahlenden Betrag (Verleihrate pro verliehenem Tag, mal Anzahl der verliehenen Tage). Wenn der bezahlte Betrag höher ist als die insgesamt zu bezahlende Verleihrate, soll folgende Fehlermeldung **ausgelöst** werden:

„Invalid amount for film <film>, maximum for <days> is <rate * days>.“

Ersetzen Sie die Elemente in der Mitteilung in der benutzerdefinierten Fehlermeldung durch die entsprechenden Werte.

Ist der angegebene Betrag kleiner als 0 wird der Betrag auf 0 korrigiert und zurückgegeben. Testen Sie die Prozedur zumindest, indem Sie für den Film 200 überprüfen, ob ein Betrag von 20 für 3 Tage möglich ist. Rufen Sie dazu die Prozedur in einem anonymen Block auf.

Hinweis:

Die Prozedur sollte keinen Exception Handler aufweisen, da in der nachfolgenden Aufgabe die Prozedur von einem Trigger aufgerufen wird und ein Behandeln einer Exception den Trigger erfolgreich beenden würde und somit die mit dem Trigger verbundene Datenbankoperation durchgeführt wird. D.h. es sollte bspw. kein Handler für ... WHEN NO_DATA_FOUND THEN ... definiert sein, wenn im SELECT kein Satz gefunden wird, da die film_id nicht vorhanden ist. Grundsätzlich würde dieser Fehler schon vom deklarativen Foreign Key Constraint behandelt werden.

- Erstellen Sie für die Tabelle PAYMENT einen Trigger mit dem Namen CHECK_AMOUNT_TRG, der bei einer INSERT- oder UPDATE-Operation in einer Zeile ausgelöst wird. Der Trigger soll die Prozedur CHECK_AMOUNT aufrufen, um die in der Aufgabe 1 definierte Regel auszuführen. Der Trigger soll die Film-Id und die Verleihdauer ermitteln (aus den Tabellen RENTAL und INVENTORY) und gemeinsam mit dem Betrag an die Prozedur übergeben. Stellen Sie sicher, dass der Trigger im Falle von UPDATE nur bei relevanten Attributen feuert.
- Testen Sie CHECK_AMOUNT_TRG anhand der folgenden Fälle und notieren Sie, was passiert: Aktualisieren Sie die Tabelle payment und setzen Sie den Betrag für die Verleihvorgänge 6500 (payment_id) auf 25 (amount), 3000 auf 1 und 1200 auf -10. Überprüfen Sie Ihre Änderungen und nehmen Sie diese dann wieder zurück (ROLLBACK).

4. Erweitern Sie die Tabelle PAYMENT und fügen Sie ein Logging-Feld hinzu (user_modified). Erstellen Sie einen Trigger LOG_PAYMENT, der die beiden Felder last_update und user_modified bei einer Änderung des Betrags aktualisiert. Testen Sie den Trigger ausführlich (führen Sie nach den Tests ein ROLLBACK aus, um Ihre Datenbasis nicht zu verändern). Anschließend entfernen Sie die zusätzliche Spalte (user_modified) wieder aus Ihrer PAYMENT-Tabelle.

2. INSTEAD OF-Trigger

(2+4+1 = 7 Punkte)

INSTEAD OF-Trigger werden ausschließlich für Sichten eingesetzt, um Daten zu ändern, bei denen eine DML-Anweisung für eine Sicht abgesetzt wird, die implizit nicht aktualisierbar ist. Diese Trigger werden INSTEAD OF-Trigger genannt, da der Datenbankserver im Gegensatz zu anderen Trigger-Typen nicht die auslösende Anweisung ausführt, sondern den Trigger auslöst. Mit diesem Trigger werden INSERT-, UPDATE- oder DELETE-Operationen direkt für die zu Grunde liegenden Basistabellen durchgeführt. Sie können INSERT-, UPDATE- oder DELETE-Anweisungen für eine Sicht erstellen, und der INSTEAD OF-Trigger arbeitet unsichtbar im Hintergrund und sorgt für die Ausführung der gewünschten Aktionen.

Führen Sie folgendes Skript aus, um die Basistabellen für die Aufgabe zu erstellen.

```
DROP TABLE new_rental;
DROP TABLE new_customer;

CREATE TABLE new_rental AS
SELECT *
FROM rental;
ALTER TABLE new_rental ADD CONSTRAINT new_rental_pk PRIMARY KEY (rental_id);

CREATE TABLE new_customer (customer_id, store_id NULL, first_name, last_name,
email, address_id NULL, active, create_date, last_update) AS
SELECT *
FROM customer;

ALTER TABLE new_customer ADD CONSTRAINT new_customer_pk PRIMARY KEY
(customer_id);
```

Skript UE05_02.sql

- Erstellen Sie eine Sicht premium_customer basierend auf den im obigen Skript erstellten Tabellen NEW_CUSTOMER und NEW_RENTAL mit folgenden Spalten: customer_id, first_name, last_name und numFilms (zählt die Anzahl der geliehenen Filme, d.h. Inventar). Es sollen nur Kunden in der View enthalten sein, die mehr als 30 Filme geliehen haben.
- Legen Sie nun für die zuvor erstellte Sicht premium_customer einen INSTEAD OF-Trigger an, um DML-Operationen auf dieser Sicht zu kontrollieren. Folgende Funktionalitäten sind gefordert:
 - Bei einem DELETE wird der Satz aus new_customer gelöscht und auch alle zugehörigen Verleihvorgänge aus new_rental.
 - Bei einem INSERT wird ein neuer Kunde in new_customer angelegt und alle vorhandenen Daten ausgefüllt (inkl. Erst- und Änderungsdatum). Wird kein Primärschlüssel beim INSERT angegeben, so stellen Sie einen automatisch bereit. Hierfür berechnen Sie vorher auf Basis der höchsten ID einen möglichen Schlüssel, den Sie verwenden.
 - Bei einem UPDATE auf numFilms geben Sie eine entsprechende Fehlermeldung zurück, dass dieses berechnete Feld nicht aktualisiert werden kann. D.h. Sie lösen hierfür einen entsprechenden selbst definierten Fehler aus.
 - Bei einem UPDATE auf customer_id geben Sie eine entsprechende Fehlermeldung zurück, dass ein Primärschlüssel nicht verändert werden darf.

- e. Bei einem UPDATE auf die anderen Felder aktualisieren Sie diese in der Basistabelle new_customer.

3. Überprüfen Sie die implementierte Funktionalität des INSTEAD OF-Triggers mit mind. den folgenden Statements.

```
INSERT INTO premium_customer (customer_id, first_name, last_name)
VALUES (3000, 'Mary', 'Poppins');
INSERT INTO premium_customer (first_name, last_name)
VALUES ('Mary', 'Poppins');

SELECT *
FROM new_customer
WHERE LOWER(last_name) LIKE 'poppins';

DELETE FROM premium_customer
WHERE customer_id = 51;

SELECT *
FROM new_customer
WHERE customer_id = 51;

SELECT *
FROM new_rental
WHERE customer_id = 51;

UPDATE premium_customer
SET numFilms = 40
WHERE customer_id = 470;

UPDATE premium_customer
SET customer_id = 40
WHERE customer_id = 470;

UPDATE premium_customer
SET first_name = 'John',
    last_name = 'Smith'
WHERE customer_id = 470;

SELECT *
FROM new_customer
WHERE customer_id = 470;
```

3. Trigger für Systemereignisse

(1+3+1 = 5 Punkte)

1. Erstellen Sie eine Tabelle USER_LOGGING mit den Feldern session_id, login_time, db_user, os_user, ip und host_name. Wählen Sie geeignete Datentypen.
2. Erstellen Sie für das Schema einen Systemtrigger, der pro Session beim Anmelden einen Datensatz einfügt. Verwenden Sie die Funktion SYS_CONTEXT mit den entsprechenden Parametern um die Session-ID, die IP-Adresse, den Betriebssystem-User und den Host-Namen (Bezeichnung des verbundenen Rechners) zu ermitteln. Schreiben Sie diese Werte gemeinsam mit dem angemeldeten Datenbank-User und eines aktuellen Zeitstempels in die Tabelle.
3. Melden Sie sich bei der Datenbank ab und wieder an. Wiederholen Sie diese Schritte (wenn möglich) auch von einem anderen Rechner aus. Zeigen Sie den Inhalt Ihrer Tabelle an.

DES3UE Übung 5

Elias Leonhardsberger

20. Dezember 2024, Hagenberg

Inhaltsverzeichnis

1	Trigger (Sakila)	5
1.1	SQL-Statements	5
1.1.1	Aufgabe 1	5
1.1.2	Aufgabe 2	6
1.1.3	Aufgabe 3	6
1.1.4	Aufgabe 4	7
1.2	Ergebnisse	8
2	INSTEAD OF-Trigger	10
2.1	SQL-Statements	10
2.1.1	Setup	10
2.1.2	Aufgabe 1	10
2.1.3	Aufgabe 2	10
2.1.4	Aufgabe 3	11
2.2	Ergebnisse	12
3	Trigger für Systemereignisse	14
3.1	SQL-Statements	14
3.1.1	Aufgabe 1	14
3.1.2	Aufgabe 2	14
3.2	Ergebnisse	14

1 Trigger (Sakila)

1.1 SQL-Statements

1.1.1 Aufgabe 1

```
1 CREATE OR REPLACE PROCEDURE CHECK_AMOUNT(RENTED_FILM_ID
  ↳ FILM.FILM_ID%TYPE, PAID_AMOUNT FILM.RENTAL_RATE%TYPE,
2                                     RENTAL_DURATION NUMBER,
                                     ↳ CORRECTED_AMOUNT OUT
                                     ↳ FILM.RENTAL_RATE%TYPE) AS
3     FILM_RENTAL_RATE FILM.RENTAL_RATE%TYPE;
4 BEGIN
5     SELECT RENTAL_RATE INTO FILM_RENTAL_RATE FROM FILM WHERE
  ↳ RENTED_FILM_ID = FILM_ID;
6
7     IF FILM_RENTAL_RATE * RENTAL_DURATION < PAID_AMOUNT THEN
8         RAISE_APPLICATION_ERROR(-20001,
9                                     'Invalid amount for film ' ||
  ↳ RENTED_FILM_ID || ', maximum for ' ||
  ↳ RENTAL_DURATION ||
10                                     ' days is ' || RENTAL_DURATION *
  ↳ FILM_RENTAL_RATE || '.');
11     END IF;
12
13     IF PAID_AMOUNT < 0 THEN
14         CORRECTED_AMOUNT := 0;
15     ELSE
16         CORRECTED_AMOUNT := PAID_AMOUNT;
17     END IF;
18 END;
19
20 -- Aufgabe 1_1
21 DECLARE
22     amount NUMBER;
23 BEGIN
24     amount := 17;
25     dbms_output.put_line('Test 200, 5, 3');
26     CHECK_AMOUNT(200, 5, 3, amount);
27     DBMS_OUTPUT.PUT_LINE('Out Amount: ' || amount);
28
29     dbms_output.put_line('Test 200, -20, 3');
30     CHECK_AMOUNT(200, -20, 3, amount);
31     DBMS_OUTPUT.PUT_LINE('Out Amount: ' || amount);
32
33     dbms_output.put_line('Test 200, 20, 3');
34     CHECK_AMOUNT(200, 20, 3, amount);
35     DBMS_OUTPUT.PUT_LINE('Out Amount: ' || amount);
```

```

36 EXCEPTION
37     WHEN OTHERS THEN
38         dbms_output.put_line('Error: ' || SQLERRM);
39 END;

```

1.1.2 Aufgabe 2

```

1  CREATE OR REPLACE TRIGGER CHECK_AMOUNT_TRG
2      BEFORE INSERT OR UPDATE OF AMOUNT
3      ON PAYMENT
4      FOR EACH ROW
5  DECLARE
6      CORRECTED_AMOUNT NUMBER;
7      RENTED_FILM_ID FILM.FILM_ID%TYPE;
8      RENTAL_DURATION NUMBER;
9
10 BEGIN
11     -- for checking if the trigger is fired
12     Dbms_output.put_line('Trigger fired');
13
14     SELECT i.FILM_ID, r.RETURN_DATE - r.RENTAL_DATE
15     INTO RENTED_FILM_ID, RENTAL_DURATION
16     FROM RENTAL r
17     INNER JOIN INVENTORY i
18         ON r.INVENTORY_ID = i.INVENTORY_ID
19     WHERE r.RENTAL_ID = :NEW.RENTAL_ID;
20
21     CHECK_AMOUNT(RENTED_FILM_ID, :NEW.AMOUNT, RENTAL_DURATION,
22         ↪ CORRECTED_AMOUNT);
23
24     :NEW.AMOUNT := CORRECTED_AMOUNT;
25 END;

```

1.1.3 Aufgabe 3

```

1  -- Ausgabe 1_3_1
2  SELECT *
3  FROM PAYMENT
4  WHERE PAYMENT_ID IN (6500, 3000, 1200);
5
6  -- fails
7  -- Ausgabe 1_3_2
8  UPDATE payment
9  SET amount = 25
10 WHERE payment_id = 6500;
11
12 -- works
13 -- Ausgabe 1_3_3

```

```

14 UPDATE payment
15 SET amount = 1
16 WHERE payment_id = 3000;
17
18 -- works
19 -- Ausgabe 1_3_4
20 UPDATE payment
21 SET amount = -10
22 WHERE payment_id = 1200;
23
24 -- works, no trigger fired
25 -- Ausgabe 1_3_5
26 UPDATE payment
27 SET PAYMENT_DATE = TO_DATE('2020-03-19', 'YYYY-MM-DD')
28 WHERE payment_id = 1200;
29
30 -- Ausgabe 1_3_6
31 SELECT * FROM PAYMENT WHERE PAYMENT_ID IN (6500, 3000, 1200);
32
33 ROLLBACK;

```

1.1.4 Aufgabe 4

```

1 ALTER TABLE payment
2     ADD user_modified VARCHAR2(50);
3
4 CREATE OR REPLACE TRIGGER LOG_PAYMENT
5     BEFORE UPDATE OF AMOUNT
6     ON PAYMENT
7     FOR EACH ROW
8 BEGIN
9     :NEW.user_modified := USER;
10    :NEW.last_update := SYSDATE;
11 END;
12
13 -- Ausgabe 1_4_1
14 SELECT *
15 FROM PAYMENT
16 WHERE PAYMENT_ID IN (6500, 3000, 1200);
17
18 -- fails, no log
19 -- Ausgabe 1_4_2
20 UPDATE payment
21 SET amount = 25
22 WHERE payment_id = 6500;
23
24 -- works, logs
25 -- Ausgabe 1_4_3

```

```

26 UPDATE payment
27 SET amount = 1
28 WHERE payment_id = 3000;
29
30 -- works, no trigger fired => no log
31 -- Ausgabe 1_4_4
32 UPDATE payment
33 SET PAYMENT_DATE = TO_DATE('2020-03-19', 'YYYY-MM-DD')
34 WHERE payment_id = 1200;
35
36 -- Ausgabe 1_4_5
37 SELECT *
38 FROM PAYMENT
39 WHERE PAYMENT_ID IN (6500, 3000, 1200);
40
41 ROLLBACK;
42
43 ALTER TABLE PAYMENT
44     DROP COLUMN user_modified;
45
46 DROP TRIGGER LOG_PAYMENT;

```

1.2 Ergebnisse

```

Test 200, 5, 3
Out Amount: 5
Test 200, -20, 3
Out Amount: 0
Test 200, 20, 3
Error: ORA-20001: Invalid amount for film 200, maximum for 3 days is 6.57.

```

Abbildung 1: Ausgabe 1_1

	PAYMENT_ID	CUSTOMER_ID	STAFF_ID	RENTAL_ID	AMOUNT	PAYMENT_DATE	LAST_UPDATE
1	1200	43	5	15945	8.83	2015-06-23	2019-07-06 21:43:57.000000
2	3000	112	2	701	1.12	2015-01-17	2019-08-14 12:26:52.000000
3	6500	241	3	627	15.93	2014-09-30	2019-06-19 16:36:51.000000

Abbildung 2: Ausgabe 1_3_1

```
S2310307019> UPDATE payment
                SET amount = 25
                WHERE payment_id = 6500
[2024-12-20 18:39:14] [72000][20001]
[2024-12-20 18:39:14] ORA-20001: Invalid amount for film 93, maximum for 7 days is 17.43.
[2024-12-20 18:39:14] ORA-06512: at "S2310307019.CHECK_AMOUNT", line 8
[2024-12-20 18:39:14] ORA-06512: at "S2310307019.CHECK_AMOUNT_TRG", line 17
[2024-12-20 18:39:14] ORA-04088: error during execution of trigger 'S2310307019.CHECK_AMOUNT_TRG'
[2024-12-20 18:39:14] Position: 7
Trigger fired
```

Abbildung 3: Ausgabe 1_3_2

```
S2310307019> UPDATE payment
                SET amount = 1
                WHERE payment_id = 3000
[2024-12-20 18:41:48] 1 row affected in 5 ms
Trigger fired
```

Abbildung 4: Ausgabe 1_3_3

```
S2310307019> UPDATE payment
                SET amount = -10
                WHERE payment_id = 1200
[2024-12-20 18:42:22] 1 row affected in 9 ms
Trigger fired
```

Abbildung 5: Ausgabe 1_3_4

```
S2310307019> UPDATE payment
                SET PAYMENT_DATE = TO_DATE('2020-03-19', 'YYYY-MM-DD')
                WHERE payment_id = 1200
[2024-12-20 18:42:42] 1 row affected in 6 ms
```

Abbildung 6: Ausgabe 1_3_5

	PAYMENT_ID	CUSTOMER_ID	STAFF_ID	RENTAL_ID	AMOUNT	PAYMENT_DATE	LAST_UPDATE
1	1200	43	5	15945	0.00	2020-03-19	2019-07-06 21:43:57.000000
2	3000	112	2	701	1.00	2015-01-17	2019-08-14 12:26:52.000000
3	6500	241	3	627	15.93	2014-09-30	2019-06-19 16:36:51.000000

Abbildung 7: Ausgabe 1_3_6

2 INSTEAD OF-Trigger

2.1 SQL-Statements

2.1.1 Setup

```
1 DROP TABLE new_rental;
2 DROP TABLE new_customer;
3
4 CREATE TABLE new_rental AS
5 SELECT *
6 FROM rental;
7 ALTER TABLE new_rental ADD CONSTRAINT new_rental_pk PRIMARY KEY
  ↪ (rental_id);
8
9 CREATE TABLE new_customer (customer_id, store_id NULL, first_name,
  ↪ last_name, email, address_id NULL, active, create_date, last_update)
  ↪ AS
10 SELECT *
11 FROM customer;
12
13 ALTER TABLE new_customer ADD CONSTRAINT new_customer_pk PRIMARY KEY
  ↪ (customer_id);
```

2.1.2 Aufgabe 1

```
1 CREATE OR REPLACE VIEW premium_customer AS
2 SELECT C.CUSTOMER_ID, C.FIRST_NAME, C.LAST_NAME, COUNT(r.RENTAL_ID) AS
  ↪ numFilms
3 FROM NEW_CUSTOMER C
4 INNER JOIN NEW_RENTAL r
5     ON C.CUSTOMER_ID = r.CUSTOMER_ID
6 GROUP BY C.CUSTOMER_ID, C.FIRST_NAME, C.LAST_NAME
7 HAVING COUNT(r.RENTAL_ID) > 30;
```

2.1.3 Aufgabe 2

```
1 CREATE OR REPLACE TRIGGER instead_of_trigger
2     INSTEAD OF INSERT OR UPDATE OR DELETE
3     ON premium_customer
4     FOR EACH ROW
5 DECLARE
6     new_customer_id NUMBER;
7 BEGIN
8     IF DELETING THEN
9         DELETE
10        FROM NEW_RENTAL
11        WHERE customer_id = :OLD.customer_id;
12        DELETE
```

```

13      FROM NEW_CUSTOMER
14      WHERE customer_id = :OLD.customer_id;
15  ELSIF INSERTING THEN
16      IF :NEW.customer_id IS NULL THEN
17          SELECT MAX(CUSTOMER_ID) + 1
18          INTO new_customer_id
19          FROM NEW_CUSTOMER;
20      ELSE
21          new_customer_id := :NEW.customer_id;
22      END IF;
23      INSERT INTO NEW_CUSTOMER (CUSTOMER_ID, FIRST_NAME, LAST_NAME,
24      ↪ CREATE_DATE, LAST_UPDATE)
25      VALUES (new_customer_id, :NEW.FIRST_NAME, :NEW.LAST_NAME,
26      ↪ SYSDATE, SYSDATE);
27  ELSIF UPDATING ('numFilms') THEN
28      RAISE_APPLICATION_ERROR(-20000, 'You cannot update the number of
29      ↪ films');
30  ELSIF UPDATING ('CUSTOMER_ID') THEN
31      RAISE_APPLICATION_ERROR(-20000, 'You cannot update the primary
32      ↪ key customer id');
33  ELSIF UPDATING THEN
34      UPDATE NEW_CUSTOMER
35      SET FIRST_NAME = :NEW.FIRST_NAME,
36      LAST_NAME = :NEW.LAST_NAME,
37      LAST_UPDATE = SYSDATE
38      WHERE CUSTOMER_ID = :NEW.CUSTOMER_ID;
39  END IF;
40 END;

```

2.1.4 Aufgabe 3

```

1  -- Ausgabe 2_3_1
2  INSERT INTO premium_customer (customer_id, first_name, last_name)
3  VALUES (3000, 'Mary', 'Poppins');
4  INSERT INTO premium_customer (first_name, last_name)
5  VALUES ('Mary', 'Poppins');
6
7  -- Ausgabe 2_3_2
8  SELECT *
9  FROM new_customer
10 WHERE LOWER(last_name) LIKE 'poppins';
11
12 DELETE
13 FROM premium_customer
14 WHERE customer_id = 51;
15
16 -- Ausgabe 2_3_3
17 SELECT *

```

```

18 FROM new_customer
19 WHERE customer_id = 51;
20
21 -- Ausgabe 2_3_4
22 SELECT *
23 FROM new_rental
24 WHERE customer_id = 51;
25
26 -- Ausgabe 2_3_5
27 UPDATE premium_customer
28 SET numFilms = 40
29 WHERE customer_id = 470;
30
31 -- Ausgabe 2_3_6
32 UPDATE premium_customer
33 SET customer_id = 40
34 WHERE customer_id = 470;
35
36 UPDATE premium_customer
37 SET first_name = 'John',
38     last_name  = 'Smith'
39 WHERE customer_id = 470;
40
41 -- Ausgabe 2_3_7
42 SELECT *
43 FROM new_customer
44 WHERE customer_id = 470;
45
46 ROLLBACK;

```

2.2 Ergebnisse

```

S2310307019> INSERT INTO premium_customer (customer_id, first_name, last_name)
              VALUES (3000, 'Mary', 'Poppins')
[2024-12-20 18:52:55] 1 row affected in 14 ms
S2310307019> INSERT INTO premium_customer (first_name, last_name)
              VALUES ('Mary', 'Poppins')
[2024-12-20 18:52:55] 1 row affected in 8 ms

```

Abbildung 8: Ausgabe 2_3_1

	CUSTOMER_ID	STORE_ID	FIRST_NAME	LAST_NAME	EMAIL	ADDRESS_ID	ACTIVE	CREATE_DATE	LAST_UPDATE
1	3000	<null>	Mary	Poppins	<null>	<null>	<null>	2024-12-20 18:52:55	2024-12-20 18:52:55.000000
2	3001	<null>	Mary	Poppins	<null>	<null>	<null>	2024-12-20 18:52:55	2024-12-20 18:52:55.000000

Abbildung 9: Ausgabe 2_3_2

	CUSTOMER_ID	STORE_ID
--	-------------	----------

Abbildung 10: Ausgabe 2_3_3

	RENTAL_ID	RENTAL_D
--	-----------	----------

Abbildung 11: Ausgabe 2_3_4

```
S2310307019> UPDATE premium_customer
               SET numFilms = 40
               WHERE customer_id = 470
[2024-12-20 18:55:56] [72000][20000]
[2024-12-20 18:55:56] ORA-20000: You cannot update the number of films
[2024-12-20 18:55:56] ORA-06512: at "S2310307019.INSTEAD_OF_TRIGGER", line 22
[2024-12-20 18:55:56] ORA-04088: error during execution of trigger 'S2310307019.INSTEAD_OF_TRIGGER'
[2024-12-20 18:55:56] Position: 7
```

Abbildung 12: Ausgabe 2_3_5

```
S2310307019> UPDATE premium_customer
               SET customer_id = 40
               WHERE customer_id = 470
[2024-12-20 18:56:17] [72000][20000]
[2024-12-20 18:56:17] ORA-20000: You cannot update the primary key customer id
[2024-12-20 18:56:17] ORA-06512: at "S2310307019.INSTEAD_OF_TRIGGER", line 24
[2024-12-20 18:56:17] ORA-04088: error during execution of trigger 'S2310307019.INSTEAD_OF_TRIGGER'
[2024-12-20 18:56:17] Position: 7
```

Abbildung 13: Ausgabe 2_3_6

	CUSTOMER_ID	STORE_ID	FIRST_NAME	LAST_NAME	EMAIL	ADDRESS_ID	ACTIVE	CREATE_DATE	LAST_UPDATE
1	470		John	Smith	GORDON.ALLARD@sakilacustomer.org		475	1 2006-02-14	2024-12-20 18:56:4

Abbildung 14: Ausgabe 2_3_7

3 Trigger für Systemereignisse

3.1 SQL-Statements

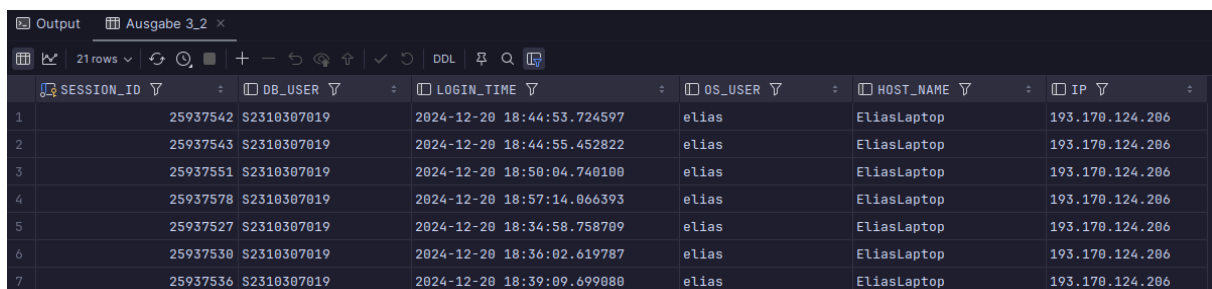
3.1.1 Aufgabe 1

```
1 CREATE TABLE USER_LOGGING
2 (
3     session_id NUMBER PRIMARY KEY,
4     db_user    VARCHAR2(30),
5     login_time TIMESTAMP,
6     os_user    VARCHAR2(30),
7     host_name  VARCHAR2(30),
8     ip        VARCHAR2(40)
9 );
```

3.1.2 Aufgabe 2

```
1
2 CREATE OR REPLACE TRIGGER log_user_session
3     AFTER LOGON
4     ON SCHEMA
5 BEGIN
6     INSERT INTO USER_LOGGING (session_id, db_user, login_time, os_user,
7     ↪ host_name, ip)
8     VALUES (SYS_CONTEXT('USERENV', 'SESSIONID'), USER, SYSTIMESTAMP,
9     ↪ SYS_CONTEXT('USERENV', 'OS_USER'),
10             SYS_CONTEXT('USERENV', 'HOST'), SYS_CONTEXT('USERENV',
11             ↪ 'IP_ADDRESS'));
12 END;
13
14 -- Ausgabe 3_2
15 SELECT * FROM USER_LOGGING;
```

3.2 Ergebnisse



The screenshot shows a database query result with 7 rows. The columns are SESSION_ID, DB_USER, LOGIN_TIME, OS_USER, HOST_NAME, and IP. The data is as follows:

	SESSION_ID	DB_USER	LOGIN_TIME	OS_USER	HOST_NAME	IP
1	25937542	S2310307019	2024-12-20 18:44:53.724597	elias	EliasLaptop	193.170.124.206
2	25937543	S2310307019	2024-12-20 18:44:55.452822	elias	EliasLaptop	193.170.124.206
3	25937551	S2310307019	2024-12-20 18:50:04.740100	elias	EliasLaptop	193.170.124.206
4	25937578	S2310307019	2024-12-20 18:57:14.066393	elias	EliasLaptop	193.170.124.206
5	25937527	S2310307019	2024-12-20 18:34:58.758709	elias	EliasLaptop	193.170.124.206
6	25937530	S2310307019	2024-12-20 18:36:02.619787	elias	EliasLaptop	193.170.124.206
7	25937536	S2310307019	2024-12-20 18:39:09.699080	elias	EliasLaptop	193.170.124.206

Abbildung 15: Ausgabe 3_2