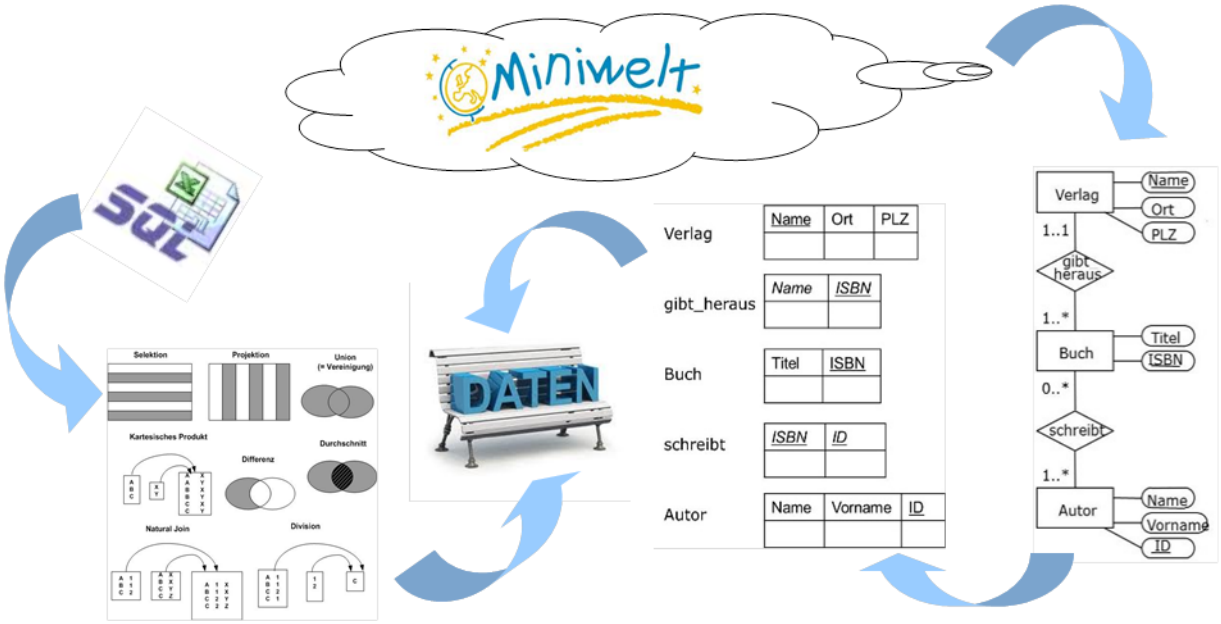
DEM2UE Datenmodellierung und Datenbankdesign SS 2021 Übung 11

Abgabetermin: 10. Juni 2021, 12 Uhr

☐ **DEM2 G1 Dr. Pitzer** **Name** Angelos Angelis **Aufwand in h** 7
☒ **DEM2 G2 Dr. Pitzer**
☐ **DEM2 G3 Dr. Niklas** **Punkte** _____ **Kurzzeichen Tutor** _____

Ziel dieser Übung ist es, die Anfragesprache SQL im Bereich Datenmanipulation (Einfügen, Löschen, Ändern von Datensätzen) praktisch zu vertiefen sowie das Sicht-Konzept anzuwenden.



1. Daten bearbeiten – INSERT, UPDATE, DELETE (Human Resources) (7 Punkte)

Die Personalabteilung möchte, dass Sie SQL-Anweisungen für das Einfügen, Aktualisieren und Löschen von Angestelltendaten erstellen. Bevor Sie die Anweisungen an die Personalabteilung übergeben, erzeugen Sie die angegebenen Tabellen `MY_EMPLOYEE` und `MY_JOB_HISTORY` als Hilfstabellen, um ihre SQL-Anweisungen zu testen.

```
CREATE TABLE MY_EMPLOYEE (
  id NUMBER(6) CONSTRAINT my_employee_id_pk PRIMARY KEY,
  last_name VARCHAR2(25) NOT NULL,
  first_name VARCHAR2(20) NOT NULL,
  userid VARCHAR2(8) NOT NULL,
  salary NUMBER(8,2) NOT NULL);

CREATE TABLE MY_JOB_HISTORY (
  id NUMBER(6) NOT NULL,
  start_date DATE NOT NULL,
  end_date DATE NOT NULL,
  job_id VARCHAR2(10) NOT NULL,
  department_id NUMBER(4) NOT NULL);
```

```
--1.
INSERT INTO my_employee VALUES(1, 'Patel', 'Ralph', 'rapatel', '895');

--2.
INSERT INTO my_employee (ID, LAST_NAME, FIRST_NAME, USERID, SALARY)
  VALUES(2, 'Dancs', 'Betty', 'bdancs', '860');

--3.
COMMIT;

INSERT ALL
  INTO my_employee VALUES(3, 'Biri', 'Ben', 'bbiri', '1100')
  INTO my_employee VALUES(4, 'Newman', 'Chad', 'cnewman', '750')
  INTO my_employee VALUES(5, 'Ropeburn', 'Audrey', 'aropebur', '1550')
SELECT * FROM dual;

--4.
UPDATE my_employee SET last_Name = 'Drexler' WHERE ID = 3;

--5.
UPDATE my_employee SET salary = '1000' WHERE salary < 900;

--6.
DELETE FROM my_employee WHERE last_name = 'Dancs' AND first_name = 'Betty';
COMMIT;

--7.
DELETE FROM my_employee;
SELECT * FROM my_employee;
```

```
--8.
ROLLBACK;
SELECT * FROM my_employee;

--9.
INSERT INTO my_job_history(id, start_date, end_date, job_id, department_id)
SELECT employee_id, start_date, end_date, job_id, department_id
  FROM job_history;

COMMIT;

DELETE FROM my_job_history h
 WHERE start_date = (SELECT MIN(start_date)
                     FROM my_job_history j
                    WHERE h.id = j.id
                   GROUP BY j.id
                   HAVING COUNT(id) > 1);
SELECT *
FROM my_job_history;

--Aufgabe 2

MERGE INTO bonus b
USING employees e
ON (b.employee_id = e.employee_id)
WHEN MATCHED THEN
   UPDATE SET b.bonus = b.bonus *1.15
      WHERE e.salary < 11000
```

```
   DELETE WHERE e.salary >= 11000
   OR e.department_id = 50
WHEN NOT MATCHED THEN
   INSERT (employee_id,bonus)
   VALUES (e.employee_id, e.salary * 0.01)
   WHERE e.department_id !=50 AND e.salary < 11000;


--AUFGABE 3


--1.
CREATE VIEW EMPLOYEES_VU AS
   SELECT employee_id AS ID, first_name || ' ' || last_name AS NAME, department_id AS
DEPARTMENT_ID
     FROM employees;
SELECT *
 FROM EMPLOYEES_VU;
```

	ID	NAME	DEPARTMENTID
1	100	Steven King	90
2	101	Neena Kochhar	90
3	102	Lex De Haan	90
4	103	Alexander Hunold	60
5	104	Bruce Ernst	60
6	107	Diana Lorentz	60
7	124	Kevin Mourgos	50
8	141	Trenna Rajs	50
9	142	Curtis Davies	50
10	143	Randall Matos	50
11	144	Peter Vargas	50
12	149	Eleni Zlotkey	80
13	174	Ellen Abel	80
14	176	Jonathon Taylor	80
15	178	Kimberely Grant	(null)
16	200	Jennifer Whalen	10
17	201	Michael Hartstein	20
18	202	Pat Fay	20

--2.

```
CREATE VIEW DEPT50 AS
   SELECT employee_id AS EMPNO, last_name AS employee, department_id as DEPTNO
    FROM employees
   WHERE department_id = 50
WITH CHECK OPTION;


SELECT *
 FROM DEPT50;
```

	EMPNO	EMPLOYEE	DEPTNO
1	124	Mourgos	50
2	141	Rajs	50
3	142	Davies	50
4	143	Matos	50
5	144	Vargas	50

--3.

```
UPDATE DEPT50
  SET DEPTNO = 80
 WHERE employee = 'Matos';
```

--Funktioniert nicht(wie gewollt): Fehlerbericht Verletzung der WHERE-Klausel einer View WITH CHECK OPTION

--4.

```
CREATE VIEW SALVU50 AS
   SELECT employee_id AS ID_NUMBER, last_name AS NAME, (salary * 12) AS ANN_SALARY
    FROM employees
   WHERE department_id = 50;
```

```
SELECT *
 FROM SALVU50;
```

	ID_NUMBER	NAME	ANN_SALARY
1	124	Mourgos	69600
2	141	Rajs	42000
3	142	Davies	37200
4	143	Matos	31200
5	144	Vargas	30000

```
--5.
CREATE VIEW DETAILEDDEP AS
   SELECT department_name AS NAME, MIN(salary) AS MINSAL, MAX(salary) AS MAXSAL, AVG(salary) AS AVGSAL
     FROM employees
   INNER JOIN departments USING (department_id)
    GROUP BY department_name;

SELECT *
 FROM DETAILEDDEP;
```

	NAME	MINSAL	MAXSAL	AVGSAL
1	Sales	8600	11000	10033,33333333333333333333333333333333
2	Marketing	6000	13000	9500
3	Administration	4400	4400	4400
4	Shipping	2500	5800	3500
5	IT	4200	9000	6400
6	Executive	17000	24000	19333,33333333333333333333333333333333
7	Accounting	8300	12000	10150

--6.

```
CREATE VIEW EMPVU10 AS
   SELECT employee_id, last_name, job_id
     FROM employees
    WHERE department_id = 10
WITH READ ONLY;
```

```
SELECT *
 FROM EMPVU10;
```

	EMPLOYEE_ID	LAST_NAME	JOB_ID
1	200	Whalen	AD_ASST

--7.

```
DELETE FROM EMPVU10
 WHERE employee_id = 10;
```

-- Aufgabe 4

--1.

Beide Abfragen also korreliert und nicht korreliert sind Unterabfragen. Der Unterschied hierbei ist, dass eine Nicht korrelierte Abfrage nur einmal ausgeführt wird während korrelierte Abfragen für jedes Ergebnistupel die Unterabfrage einmal ausgeführt wird.

--2.

```
 SELECT e.last_name, e.first_name, e.job_id, d.department_name
  FROM employees e JOIN departments d ON(e.department_id = d.department_id)
     JOIN (SELECT job_id, AVG(salary) AS avg_salary
          FROM employees
          GROUP BY job_id) salaries
     ON e.job_id = salaries.job_id
  WHERE e.salary > salaries.avg_salary ;
```

	LAST_NAME	FIRST_NAME	JOB_ID	DEPARTMENT_NAME
1	Rajs	Trenna	ST_CLERK	Shipping
2	Davies	Curtis	ST_CLERK	Shipping
3	Hunold	Alexander	IT_PROG	IT
4	Abel	Ellen	SA_REP	Sales

5. Theoriefragen (3 Punkte)

Kreuzen Sie alle der folgenden Aussagen an, die wahr sind. Bitte beachten Sie, dass Sie dazu möglicherweise auch außerhalb der zur Verfügung gestellten Unterlagen recherchieren müssen.

- [x] Virtuelle Sichten werden für die Performance-Steigerung von Abfragen eingesetzt.
- [ ] Mit der `WITH CHECK OPTION` werden vorhandene Check-Klausel der Basistabellen aktiviert.
- [x] DML Operationen sind nur unter bestimmten Bedingungen auf eine virtuelle Sicht möglich.
- [x] Die Ergebnismenge einer virtuellen Sicht wird bei Ausführung erstellt und nicht dauerhaft persistiert.
- [x] Beim Einfügen von Daten werden vorher angelegte Constraints geprüft.
- [x] Datenänderungen werden mit `COMMIT` dauerhaft festgeschrieben.
- [ ] Der Einsatz von `MERGE` ist vor allem dann sinnvoll, wenn sich Relationen nicht in der zweiten Normalform befinden.
- [x] Bei `TRUNCATE` dürfen keine aktiven `FOREIGN KEY` Constraints auf der Tabelle vorliegen.
- [x] DML Statements können abwechselnd mit einzelnen SQL Abfragen abgesetzt werden.
- [ ] Mehrere DML Statements werden vor der Ausführung vom SQL Optimizer in die richtige Reihenfolge gebracht.
- [ ] Wurden Daten bereits mit `COMMIT` festgeschrieben, können sie mit einem `SAVEPOINT` zurückgeholt werden.
- [x] Eine *Flashback Query* von Oracle erlaubt Abfragen auf einen bereits vergangenen Datenstand (Historie).