DES3UE Datenbanksysteme WS 2018 Übung 10

Abgabetermin: 12.12.2018, 13:30 Uhr

☐ **DES31UE Niklas** **Name** Niklas Veit **Aufwand in h** 10
☐ **DES32UE Niklas**
☒ **DES33UE Traxler** **Punkte** _____ **Kurzzeichen Tutor** _____

Hinweis:

Bilden Sie für die Ausarbeitung der Beispiele, insbesondere Beispiel 2.1, Gruppen und geben Sie in der Lösung den entsprechenden Verweis auf Ihren Partner/Ihre Partnerin an.

1. Zeitstempelverfahren (2,5 Punkte)

Überprüfen Sie die Serialisierbarkeit der in Abbildung 1 angegebenen Ausführung der Transaktionen T1, T2 und T3 mit Hilfe des (*nicht* optimierten) Zeitstempel-Verfahrens.

T1	T2	T3
		READ a
READ a		
WRITE b		
	WRITE d	
		WRITE d
READ c		
WRITE a		
	WRITE c	
		WRITE c
		READ b

Abbildung 1: Ausführung T1, T2 und T3

Welche Transaktionen werden abgebrochen, wenn den Transaktionen T1, T2 und T3 die Zeitstempel 75, 85 und 100 zugewiesen werden? Stellen Sie eine entsprechende Tabelle auf.

2. Vergabe und Entzug von Rechten (2 Punkte)

1. Gewähren Sie einem anderen Benutzer lesenden Zugriff auf Ihre Tabelle `DEPARTMENTS`. Lassen Sie sich von diesem Benutzer das Privileg zur Abfrage seiner Tabelle `DEPARTMENTS` erteilen und testen Sie den Zugriff.
2. Fragen Sie die View `USER_TABLES` und die View `ALL_TABLES` im Data Dictionary ab, um Informationen über die Tabellen anzuzeigen, die Ihnen gehören und auf die Sie zugreifen können. Bei der Abfrage auf die View `ALL_TABLES` schließen Sie die Tabellen aus, die Ihnen gehören.

3. COMMIT, SAVEPOINT und ROLLBACK (2,5 Punkte)

1. Erstellen Sie eine Tabelle MY_EMPLOYEE als Kopie der Tabelle employees, verwenden Sie dazu einen CREATE … AS SELECT … Befehl.
2. Erhöhen Sie das Gehalt aller Manager um 5 % (MGR und MAN). Schreiben Sie die veränderten Daten dauerhaft fest.

3. Ändern Sie das Gehalt aller Angestellten, deren Gehalt unter $ 5000 liegt, in $ 6100. Markieren Sie einen Zwischenpunkt in der Verarbeitung der Transaktion.
4. Leeren Sie die gesamte Tabelle und prüfen Sie ob die Tabelle leer ist.
5. Verwerfen Sie die letzte DELETE-Operation, ohne die vorherige UPDATE-Operationen zu verwerfen; prüfen Sie, ob die Änderungen aus Punkt 3.4 weiterhin vorhanden sind und schreiben Sie die Änderungen fest.

4. Isolationsstufen in Oracle I (Read Committed) (8 Punkte – 3+3+2)

1. In der Vorlesung wurde das Problem der verlorengegangenen Änderung (*lost update*) vorgestellt. Untersuchen Sie anhand eines selbstgewählten Schedules, ob dieses Problem auch in Oracle mit der Default-Einstellung der Isolationsstufe `READ COMMITTED` möglich ist. Verwenden Sie dazu das SQL-Skript ue10_schema.sql (enthält Beispielschemadefinition und Beispieldatensätze). Was kann ein Anwendungs- bzw. SQL-Programmierer (mit dieser Einstellung der Isolationsstufe) tun, um *lost updates* zu vermeiden? Begründen Sie Ihre Antwort.

2. Re-Initialisieren Sie Ihre Datenbank auf Basis des SQL-Skripts ue10_schema.sql und betrachten Sie die beiden folgenden Transaktionen T1 und T2 (mit der Default Isolationsstufe `READ COMMITTED`):

```
T1: Select * from emp where sal > 2800;
T1: Select * from dept where dname = 'RESEARCH';
T2: Select * from dept where dname = 'RESEARCH';
T2: Select * from emp where sal > 2800;
T1: Update emp set sal = sal*1.05 where sal > 2800;
T2: Update dept set dname = 'MARKETING' where dname = 'RESEARCH';
T2: Update emp set sal = sal*1.05 where sal > 2800;
T1: Update dept set dname = 'MARKETING' where dname = 'RESEARCH';
```

   Was stellen Sie bei diesem Ablauf fest? Was würde der Einsatz eines `SELECT ... FOR UPDATE` bei allen Select-Statements bewirken? Unter welchen Umständen können dadurch Deadlocks vermieden werden? Konstruieren Sie einen Fall (auf Basis des gegebenen Schedules) mit `SELECT ... FOR UPDATE`, in dem die Vermeidung des Deadlocks nicht möglich ist!

   Gibt es Möglichkeiten, Deadlocks in *jedem* Fall zu verhindern? Wenn ja, zählen Sie diese auf!

3. Ein weiteres Problem stellt das Non-Repeatable Read dar. Kann in Oracle bei der Isolationsstufe `READ COMMITTED` ein Non-Repeatable Read auftreten? Falls ja, geben Sie einen entsprechenden Beispiel-Schedule an und zeigen Sie Möglichkeiten auf, wie dies verhindert werden kann.

5. Isolationsstufen in Oracle II (Serializable) (9 Punkte – 3+3+3)

In der Isolationsstufe `SERIALIZABLE` verwendet Oracle das in der Vorlesung vorgestellte Mehrversionen-Concurrency-Control-Protokoll (mit der Bezeichnung Snapshot Isolation). Jede Transaktion arbeitet auf einer eigenen Version der Datenbank (snapshot), die den konsistenten Zustand aller committeter Daten zum Zeitpunkt des Transaktionsbeginns reflektiert. Leseaktionen einer Transaktion lesen also Werte so wie sie zu Beginn der Transaktion vorlagen (außer natürlich Werte, welche von dieser Transaktion selbst geschrieben wurden). Beim Schreiben wird eine neue Version des jeweiligen Objekts erzeugt, das bis zum erfolgreichen Commit nur für die eigene

Transaktion sichtbar ist.

1. Wiederholen Sie das Beispiel mit dem *lost update* Problem aus Aufgabe 4.1 mit der Isolationsstufe `SERIALIZABLE`. Kann hier das Problem des *lost updates* ebenso auftreten? Begründen Sie die sich ergebende Ausgabe!

2. Setzen Sie die Isolationsstufe auf `SERIALIZABLE` und testen Sie einen *nicht* serialisierbaren Ablauf, bei dem zwei parallele Transaktionen schreibend auf *dasselbe Objekt* zugreifen. Geben Sie die Ausgabe von Oracle an und begründen Sie diese!

3. Die Isolationsstufe `SERIALIZABLE` entspricht NICHT der Isolationsstufe `SERIALIZABLE` des SQL-2003 Standards, d. h. es können unter gewissen Bedingungen nicht serialisierbare Transaktionen fehlerfrei ausgeführt werden. Führen Sie eine Suche im Web durch, um entsprechende Anomalien herauszufinden, die in Snapshot Isolation auftreten können und beschreiben Sie diese kurz (mit jeweils einem kleinen Beispiel).

Ausarbeitung UE10

(Ausarbeitung zu 1 ganz unten aufgrund von Ausarbeitungsmethode!)

2. Vergabe und Entzug von Rechten

```
-- 2.1 mit Johann Hoffmann
GRANT SELECT ON departments TO s1710307108;
COMMIT;

SELECT *
FROM s1710307108.departments;
```

	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
1	9999	Dept 9999	<null>	<null>
2	10	Administration	200	1700
3	20	Marketing	201	1800
4	50	Shipping	124	1500
5	60	IT	103	1400
6	80	Sales	149	2500
7	90	Executive	100	1700
8	110	Accounting	205	1700
9	190	Contracting	<null>	1700

```
-- 2.2
SELECT table_name
FROM user_tables;
```

	TABLE_NAME
1	COUNTRIES
2	DEPARTMENTS
3	DEPT
4	EMP
5	EMPLOYEES
6	JOBS
7	JOB_GRADES
8	JOB_HISTORY
9	LOCATIONS
10	MY_EMPLOYEE
11	REGIONS
12	TEST

```
SELECT table_name, owner, privilege
FROM all_tables
       INNER JOIN all_tab_privs USING (table_name)
WHERE owner <> 'S1710307099'
  AND privilege = 'SELECT';
```

	TABLE_NAME	OWNER	PRIVILEGE
116	WWV_FLOW_DUAL100	APEX_040200	SELECT
117	TEST	S1710307108	SELECT
118	TEST	S1710307108	SELECT
119	DEPARTMENTS	S1710307108	SELECT
120	DEPARTMENTS	S1710307108	SELECT
121	MAP_OBJECT	SYS	SELECT
122	PSTUBTBL	SYS	SELECT
123	WRI$_ADV_ASA_RE…	SYS	SELECT
124	WRI$_HEATMAP_TO…	SYS	SELECT

3. COMMIT, SAVEPOINT und ROLLBACK

```sql
-- 3.1
CREATE TABLE my_employee
AS
SELECT *
FROM employees;

-- 3.2
-- pre update
SELECT *
FROM my_employee
WHERE job_id LIKE '%MGR'
   OR job_id LIKE '%MAN';
```

	employee_id	first_name	last_name	email	phone_number	hire_date	job_id	salary
1	124	Kevin	Mourgos	KMOURGOS	650.123.5234	1999-11-16 00:00:00	ST_MAN	5800.00
2	149	Eleni	Zlotkey	EZLOTKEY	011.44.1344.429018	2000-01-29 00:00:00	SA_MAN	10500.00
3	201	Michael	Hartstein	MHARTSTE	515.123.5555	1996-02-17 00:00:00	MK_MAN	13000.00
4	205	Shelley	Higgins	SHIGGINS	515.123.8080	1994-06-07 00:00:00	AC_MGR	12000.00

```sql
UPDATE my_employee
SET salary = salary * 1.05
WHERE job_id LIKE '%MGR'
   OR job_id LIKE '%MAN';

-- post update
SELECT *
FROM my_employee
WHERE job_id LIKE '%MGR'
   OR job_id LIKE '%MAN';
```

	employee_id	first_name	last_name	email	phone_number	hire_date	job_id	salary
1	124	Kevin	Mourgos	KMOURGOS	650.123.5234	1999-11-16 00:00:00	ST_MAN	6090.00
2	149	Eleni	Zlotkey	EZLOTKEY	011.44.1344.429018	2000-01-29 00:00:00	SA_MAN	11025.00
3	201	Michael	Hartstein	MHARTSTE	515.123.5555	1996-02-17 00:00:00	MK_MAN	13650.00
4	205	Shelley	Higgins	SHIGGINS	515.123.8080	1994-06-07 00:00:00	AC_MGR	12600.00

```sql
COMMIT;

-- 3.3
UPDATE my_employee
SET salary = 6100
WHERE salary <= 5000;

-- should yield empty table
SELECT *
FROM my_employee
WHERE salary <= 5000;
```

employee_id	first_name	last_name	email	phone_

```sql
SAVEPOINT after_salary_raise;

-- 3.4
DELETE
FROM my_employee;

SELECT *
FROM my_employee;
```

employee_id	first_name	last_name	email	phone_

```
-- 3.5
ROLLBACK TO after_salary_raise;

SELECT *
FROM my_employee;
```

	employee_id	first_name	last_name	email	phone_n
1	100	Steven	King	SKING	515.123.456
2	101	Neena	Kochhar	NKOCHHAR	515.123.456
3	102	Lex	De Haan	LDEHAAN	515.123.456
4	103	Alexander	Hunold	AHUNOLD	590.423.456
5	104	Bruce	Ernst	BERNST	590.423.456
6	107	Diana	Lorentz	DLORENTZ	590.423.556
7	124	Kevin	Mourgos	KMOURGOS	650.123.523
8	141	Trenna	Rajs	TRAJS	650.121.800

```
COMMIT; -- updates only
```

4. Isolationsstufen in Oracle I (Read Committed)

4.1.

Beispiel mit zwei Transaktionen:

```
-- Session 1
UPDATE emp
SET sal = 6000
WHERE empno = 7839;

-- Session 2 -- muss commit von S1 abwarten
UPDATE emp
SET sal = 7000
WHERE empno = 7839;

-- Session 1
SELECT *
FROM emp
WHERE empno = 7839; -- liest sal = 6000 (Wert von S1)

COMMIT; -- erst jetzt darf S2 schreiben
```

Oracle verhindert Lost-Update Fehler durch die Read Committed Isolationsstufe! Wenn zwei Transaktionen versuchen, auf dasselbe Datenobjekt zu schreiben, wird eine der zwei Transaktionen warten müssen bis die andere den Datensatz (mittels Beendung d. Transakt.) freigibt.

Anmerkung: Ich habe mir von Kollegen sagen lassen, dass es hier zu lost updates kommen kann, konnte aber keinen Ausführungsplan erzeugen, der diesen Fehler verursacht. Ich verwende Datenbank-Tools von Jetbrains und habe sichergestellt, dass ich in zwei verschiedenen Sessions unterwegs bin und mich im Read Committed Isolationsmodus befinde. Ich habe auch im Internet recherchiert und dort ergab sich ebenfalls, dass Oracle ohne Locks etc. solche Fehler nicht verhindert. Dennoch konnte ich empirisch nie ein anderes Verhalten als das oben Beschriebene feststellen.

4.2.

Beim Ausführen der angeführten Operationen entsteht ein Deadlock durch die letzte Update-Anweisung der zweiten Transaktion:

```
sql> UPDATE emp
     SET sal = sal * 1.05
     WHERE sal > 2800
[2018-12-11 22:50:30] [61000][60] ORA-00060: deadlock detected while waiting for resource
```

`FOR UPDATE` würde schon vorab notwendige Locks beantragen, was ein gegenseitiges Blockieren verhindert.

4.3.

Non-Repeatable Reads können sogar in der Isolationsstufe SERIALIZABLE (/SNAPSHOT) auftreten, siehe 5.3. Um dies zu verhinden können z.B. explizite Sperren vom Benutzer gesetzt werden.

5. Isolationsstufen in Oracle II (Serializable)

1. Durch Änderung der Isolationsstufe bricht Oracle die zweite schreibende Transaktion ab:

   ```
   sql> UPDATE emp
        SET sal = 7000
        WHERE empno = 7839
   [2018-12-12 08:57:12] [72000][8177] ORA-08177: can't serialize access for this transaction
   ```

2. Lost-Update-Fehler treten bei Schreibperationen auf das selbe Objekt auf, daher siehe 1.

3. Es können zum Beispiel dirty reads auftreten: Beide Transaktionen schreiben unterschiedliche Zeilen einer Tabelle, haben jeweils für sich konsistente Daten, können diese aber nach einem Commit nicht mehr mit gleichem Ergebnis abfragen weil die Änderungen an der Tabelle zusammengeführt wurden (Quelle: dbi-services.com):

   ```
   23:18:40 SID=365> set transaction isolation level serializable;
   Transaction set.

   23:18:41 SID=365> select sum(sal) from EMP where deptno=10;

     SUM(SAL)
   ----------
         8750

   23:18:44 SID=365> update EMP set sal=sal+250 where ename='MILLER';
   1 row updated.
   ```

```
23:18:30 SID=12> set transaction isolation level serializable;
Transaction set.

23:18:51 SID=12> select sum(sal) from EMP where deptno=10;

  SUM(SAL)
----------
      8750

23:18:53 SID=12> update EMP set sal=sal+250 where ename='CLARK';
1 row updated.
```

```
23:18:46 SID=365> select sum(sal) from EMP where deptno=10;

  SUM(SAL)
----------
      9000

23:19:04 SID=365> commit;
Commit complete.
```

```
23:18:55 SID=12> select sum(sal) from EMP where deptno=10;

  SUM(SAL)
----------
      9000

23:19:08 SID=12> commit;
Commit complete.
```

Zusammengeführte Daten nach dem COMMIT:

```
23:19:09 SID=12> select sum(sal) from EMP where deptno=10;

  SUM(SAL)
----------
      9250
```

1) Zeitstempelverfahren

(leere Felder = 0)

$t_s = 75$	$t_s = 85$	$t_s = 100$
T1	**T2**	**T3**
		READ a
READ a		
WRITE b		
	WRITE d	
		WRITE d
READ c		
↯ WRITE a		
	WRITE c	
		WRITE c
		READ b

A		B		C		D	
r	w	r	w	r	w	r	w
100							
100							
100			75				
100			75				85
100			75				100
100			75	75			100
100	↯		75	75			100

T_1 muss zurückgesetzt werden!

Anmerkung: Dr. Altmanns Folien führten den Zeitstempel der abgebrochenen Trans. weiter während die Lösungen aus der Übung nach dem ersten Konflikt komplett abbrachen. Ich wusste dementsprechend nicht, wie mit Konflikten umzugehen ist !!

T_1	T_2	T_3	A		B		C	
			$t_r(A)$	$t_w(A)$	$t_r(B)$	$t_w(B)$	$t_r(C)$	$t_w(C)$
$t_s = 200$	$t_s = 150$	$t_s = 175$	0	0	0	0	0	0
read B			0	0	200	0	0	0
	read A		150	0	200	0	0	0
		read C	150	0	200	0	175	0
write B			150	0	200	200	175	0
write A			150	200	200	200	175 ↯	0 ↯
	write C ↯		150	200	200	200	175	0
	abort		150	200	200	200	175	0
		write A	150 ✓	200 ↯	200	200	175	0

?..

Zeitstempelverfahren - Lösung

100	200	300	x		y		z	
T1	T2	T3	T R(x)	T W(x)	T R(y)	T W(y)	T R(z)	T W(z)
R(x)			100					
	R(y)		100		200			
W(x)			100	100	200			
		R(z)	100	100	200		300	
	W(x)		100	200	200		300	
		R(x)	300	200	200		300	
	W(z)		300	200	200		**300**	**200**
		W(z)						
c								
	c							
		c						