Datenbank Architektur für Fortgeschrittene

Ausarbeitung 2: Zugriffssteuerung und Views

Daniel Gürber
Stefan Eggenschwiler

28.06.2013

Inhaltsverzeichnis

1 Oracle 11g

1.1 Vorbereitung

Dieser Teil der Ausarbeitung wurde auf der uns zur Verfügung gestellten Oracle-Instanz dbarc03 durchgeführt.

1.1.1 Einrichten User

Als System:

```
1 CREATE USER nutzer01 IDENTIFIED BY nutzer01;
2 CREATE USER nutzer02 IDENTIFIED BY nutzer02;
3
4 GRANT CREATE SESSION TO nutzer01;
5 GRANT CREATE SESSION TO nutzer02;
6
7 GRANT CREATE ROLE TO scott;
8 GRANT CREATE VIEW TO scott;
```

Die User nutzer01 und nutzer02 werden angelegt und erhalten CREATE SESSION Privilegien, damit sie sich auf der Instanz anmelden können. Der User scott erhält die Privilegien Rollen und Views zu erzeugen, welche für die folgenden Schritte notwendig sind.

1.2 Zugriffssteuerung mit User und Rollen

1.2.1 Tabellen erzeugen

Als scott:

```
1 DROP TABLE klassen;
2 DROP TABLE studenten;
3
4 CREATE TABLE klassen(
5 k_id number(9),
6 k_bezeichnung VARCHAR2(20),
7 k_zimmer VARCHAR2(10),
8 k_server VARCHAR2(10));
9
10 CREATE TABLE studenten(
11 s_id number(9),
12 s_name VARCHAR2(10),
13 s_vname VARCHAR2(10),
14 s_tel VARCHAR2(20),
15 s_konto_stand NUMBER(9 ),
16 s_klasse NUMBER(9));
17
18
19 INSERT INTO klassen values ( 10, 'ia00', '3333' , 'pluto');
20 INSERT INTO klassen values ( 20, 'ia01', '2222' , 'saturn');
21
22 INSERT INTO studenten values( 101, 'meier', 'hans', '11111', 5000, 10);
23 INSERT INTO studenten values( 102, 'hirt', 'otto', '22222', -100, 10);
24 INSERT INTO studenten values( 103, 'kok', 'thomas', '33333', 1000, 20);
25 INSERT INTO studenten values( 104, 'guzman', 'anna', '44444', 3000, 20);
```

```
26 INSERT INTO studenten values( 105, 'lorch', 'felix', '45678', 7000, 20);
```

Die Tabellen klassen und studenten werden neu erstellt und mit klar definierten Startwerten gefüllt, damit die Auswirkung nachfolgender Statements nachvollziehbar sind.

1.2.2 View erstellen

Als scott:

```
1 CREATE VIEW view1 AS
2 select s_name, s_vname, k_zimmer, k_server
3 FROM studenten, klassen
4 WHERE s_klasse = k_id
```

Scott erstellt eine View view1 die daten aus den Tabellen klassen und studenten anzeigt.

1.2.3 Rollen definieren

Als scott:

```
1 CREATE ROLE view1_verwalter;
2 CREATE ROLE view1_nutzer;
```

Scott erstellt die neuen Rollen view1_verwalter und view1_nutzer.

1.2.4 Den Rollen Rechte zuweisen

Als scott:

```
1 GRANT INSERT,SELECT,UPDATE,DELETE ON studenten TO view1_verwalter;
2 GRANT INSERT,SELECT,UPDATE,DELETE ON klassen TO view1_verwalter;
```

Scott erteilt der Rolle view1_verwalter die Rechte die Daten in den Tabellen studenten und klassen anzusehen und beliebig zu verändern.

Als scott:

```
1 GRANT SELECT ON view1 TO view1_nutzer;
```

Scott erteilt der Rolle view1_nutzer das Recht die Daten von view1 anzusehen.

1.2.5 Den User Rollen zuweisen

Als scott:

```
1 GRANT view1_verwalter TO nutzer01;
2 GRANT view1_nutzer TO nutzer02;
```

Scott weist nutzer01 die Rolle view1_verwalter und nutzer02 die Rolle view1_nutzer zu.

1.2.6 Überprüfung der Rechte der beiden Rollen

Die folgenden Statements werden zur Überprüfung der Rechte verwendet:

```
1  SELECT * FROM scott.studenten;
2  INSERT INTO scott.studenten VALUES (106, 'peter', 'muster', 12345,2000,20);
3  UPDATE scott.studenten SET s_tel = 54321 WHERE s_id = 106;
4  DELETE FROM scott.studenten WHERE s_id = 106;
5
6  SELECT * FROM scott.klassen;
7  INSERT INTO scott.klassen VALUES (30, 'ia02', 1111, 'uranus');
8  UPDATE scott.klassen SET k_zimmer = 4444 WHERE k_id = 30;
9  DELETE FROM scott.klassen WHERE k_id = 30;
10
11 SELECT * FROM scott.view1;
12 INSERT INTO scott.view1 VALUES ('peter','muster',2222,'saturn');
13 UPDATE scott.view1 SET s_name = 'kook' WHERE s_vname = 'thomas';
14 DELETE FROM scott.view1 WHERE s_vname = 'thomas';
```

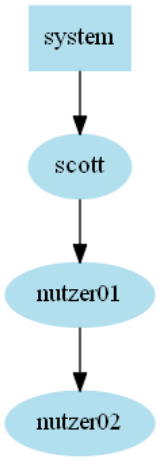
Was kann der Verwalter lesen und bearbeiten?

Die Statements in den Linien 1 bis 9 können von nutzer01 ausgeführt werden. Wenn nutzer01 versucht die Statements in den Linien 11 bis 14 auszuführen, erhält er folgende Fehlermeldung:
ORA-00942: Tabelle oder View nicht vorhanden
Dies zeigt, dass nutzer01, obwohl er Rechte auf beide von der View verwendeten Tabellen hat, keinerlei Rechte auf eine View erhält, auch wenn sie nur diese 2 Tabellen anzeigt, und somit ein Statement ausführt, welches nutzer01 selber ausführen könnte.

Was kann der Nutzer lesen und bearbeiten?

Die Statements in den Linien 1 bis 9 können von nutzer02 nicht ausgeführt werden sondern brechen mit folgender Fehlermeldung ab:
ORA-00942: Tabelle oder View nicht vorhanden
Dies zeigt, dass nutzer02 keine Zugriffsrechte auf studenten und klassen hat. Das Statement auf Linie 11 kann ausgeführt werden, wodurch gezeigt wird das nutzer02 SELECT Rechte auf view1 hat, und somit Views gut dazu geeignet sind Usern nur bestimmte Spalten oder Zeilen von Tabellen anzuzeigen. Die Statements auf den Linien 11 bis 13 brechen mit folgender Meldung ab:
ORA-01031: Nicht ausreichende Berechtigungen
Dies zeigt, dass nutzer02 view1 zwar sehen kann, aber keine Rechte hat die Daten zu ändern.

1.3 Zugriffsrechte: Objekt- und Systemrechte



1.3.1 Objektrechte

Als scott:

```
1 CREATE VIEW view2 AS
2 select s_name, s_vname, k_zimmer, k_server
3 FROM studenten, klassen
4 WHERE s_klasse = k_id;
5
6 GRANT SELECT ON view2 TO nutzer01 WITH GRANT OPTION;
```

Scott erstellt eine neue View view2 und erteilt nutzer01 SELECT-Rechte auf diese View, mit der Option diese Rechte weiterzugeben.

Als nutzer01:

```
1 GRANT SELECT ON scott view2 TO nutzer02;
```

Nutzer01 gibt das SELECT-Recht an nutzer02 weiter.

Als nutzer02:

```
1 SELECT * FROM scott view2;
```

Nutzer02 kann die Daten aus view2 anzeigen.

Als scott:

```
1 REVOKE SELECT ON view2 FROM nutzer01;
```

Scott entzieht nutzer01 die Rechte wieder.

Als nutzer02:

```
1 SELECT * FROM scott view2;
```

Das Statement schlägt mit folgender Fehlermeldung fehl:
ORA-00942: Tabelle oder View nicht vorhanden
Dies zeigt, das die Rechte die nutzer02 von nutzer01 erhalten hat, mit dem Entziehen der Rechte von nutzer01 durch scott kaskadierend weiter entzogend wurden.

1.3.2 Systemrechte

Als system:

```
1 REVOKE create session FROM nutzer01;
2 REVOKE create session FROM nutzer02;
```

Um die Vergabe von Systemrechten zu zeigen wird nutzer01 und nutzer02 das CREATE SESSION Recht entzogen. Wenn sie sich anmelden wollen erscheint folgende Fehlermeldung:
ORA-01045: user STUDENT02 lacks CREATE SESSION privilege; logon denied.

Als system:

```
1 GRANT create session TO nutzer01 WITH ADMIN OPTION;
```

Nutzer01 erhält das Recht wieder, diesmal mit ADMIN OPTION, damit er dieses weitergeben kann.

Als nutzer01:

```
1 GRANT create session TO nutzer02;
```

Nutzer01 gibt das Recht an nutzer02 weiter, jetzt können sich beide Benutzer wieder anmelden.

Als system:

```
1 REVOKE create session FROM nutzer01;
```

System einzieht nutzer01 das Recht wieder. Nutzer01 kann sich jetz nicht mehr anmelden, nutzer02 hingegen immer noch. Dies zeigt, dass Systemrechte, welche mit ADMIN OPTION weitergegeben wurden, nicht kaskadierend entzogen werden.

1.4 Views

1.4.1 Rechte von Views

Als system:

```
1 GRANT CREATE VIEW TO nutzer01;
```

Nutzer01 erhält Rechte Views zu erstellen, damit versucht werden kann, seine SELECT-Rechte über Views weiterzugeben.

Als nutzer01:

```
1 CREATE VIEW cheat_view AS
2 SELECT * FROM scott.studenten
```

Nutzer01 versucht eine View auf die geschützte Tabelle zu erstellen, dieses Statement bricht mit folgender Begründung ab:
ORA-01031: Nicht ausreichende Berechtigungen

Als system:

```
1 GRANT CREATE ANY VIEW TO nutzer01;
```

Nutzer01 erhält die Rechte Views auch auf fremde Tabellen zu erstellen und versucht nun, die View zu erstellen, dies bricht mit folgender Meldung ab:
ORA-01031: Nicht ausreichende Berechtigungen

Als system:

```
1 GRANT INSERT,SELECT,UPDATE,DELETE ON scott.studenten TO nutzer01;
```

Nutzer01 erhält die Rechte nun direkt, da ihm die Rollen-Rechte nicht erlauben Views zu erstellen, er erstellt die View nun erfolgreich.

Als nutzer01:

```
1 GRANT SELECT ON cheat_view TO nutzer02;
```

Nutzer01 gibt nun die SELECT-Rechte auf cheat_view an nutzer02.

Als nutzer02:

```
1 SELECT * FROM cheat_view
```

Nutzer02 versucht nun die Daten aus der View anzuzeigen, was mit folgender Meldung abbricht:
ORA-01720: Berechtigungsoptionen für 'SCOTT.STUDENTEN' nicht vorhanden
Views können also nicht verwendet werden um Rechte weiterzugeben, da diese beim ausführen der View wieder überprüft werden.

1.4.2 DDL-Änderungen an den Basistabellen

Alle Statements in diesem Absatz werden von scott durchgeführt.

```
1 ALTER TABLE studenten ADD birthday TIMESTAMP;
```

Scott fügt eine neue Spalte birthday zu studenten hinzu.

```
1 SELECT * FROM view1;
```

View1 kann immer noch angezeigt werden, und zeigt die gleichen Daten wie vorher an, da birthday keine der verwendeten Spalten ist.

```
1 ALTER TABLE studenten DROP COLUMN birthday;
```

Scott entfernt die birthday Spalte wieder.

```
1 ALTER TABLE studenten DROP COLUMN s_name;
```

Scott entfernt die s_name spalte.

```
1 SELECT * FROM view1;
```

View1 kann nun nicht mehr angezeigt werden, sondern das Statement bricht mit folgender Begründung ab:
ORA-04063: view "view1enthält Fehler
Dies zeigt das Views nur von Änderungen an von ihnen verwendeten Spalten betroffen sind, da view1 s_name verwendet.

```
ALTER TABLE studenten ADD s_name VARCHAR2(10);
```

Scott fügt s_name wieder ein, view1 kann nun wieder angezeigt werden, enthält in s_name nun allerdings den wert NULL.

1.4.3 Updatable Views

Alle Statements in diesem Absatz werden von scott durchgeführt.

```
CREATE VIEW view_distinct AS
SELECT DISTINCT s_vname
FROM studenten;

UPDATE view_distinct SET s_vname='ana' WHERE s_vname='anna';
```

Scott erstellt eine View die alle vorkommenden Vornamen der Studenten genau einmal anzeigt. Beim Versuch das Update-Statement auszuführen erhält scott folgende Meldung:
ORA-01732: Datenmanipulationsoperation auf dieser View nicht zulässig
Dies zeigt, das wie spezifiziert keine Daten auf Views geändert werden können die DISTINCT oder GROUP BY verwenden.

```
CREATE VIEW view_all AS
SELECT *
FROM studenten
INNER JOIN klassen ON s_klassen=k_id;

UPDATE view_all SET s_name='muster' WHERE s_id=101;
```

Scott erstellt eine View, die alle Studenten mit den Informationen zu ihren klassen anzeigt. Er versucht nun mit einem Update-Statement den Namen vom Student mit der ID 101 auf 'muster' zu ändern, dies schlägt mit folgender Begründung fehl:
ORA-01779: Kann keine Spalte, die einer Basistabelle zugewiesen wird, verändern
Bei Oracle sind Daten in Views nur änderbar wenn der Schlüssel in der View auch ein Schlüssel auf der Tabelle ist. S_id ist nicht als Schlüssel definiert.

```
ALTER TABLE studenten
ADD CONSTRAINT STUDENT_PK PRIMARY KEY ( S_ID ) ENABLE;

ALTER TABLE klassen
ADD CONSTRAINT KLASSE_PK PRIMARY KEY ( K_ID ) ENABLE;
```

S_id und k_id werden als Primärschlüssel definiert, damit die Daten einen Schlüssel haben.

```
UPDATE view_all SET s_name='muster' WHERE s_id=101;
```

Das Update kann nun erfolgreich durchgeführt werden, da s_id sowohl ein schlüssel auf view_all und studenten ist.

```
UPDATE view_all SET k_bezeichnung='test' WHERE k_id=10;
```

Dieses Update-Statement versucht nun die Bezeichnung der Klasse mit ID 10 auf 'test' zu ändern, es schlägt mit folgender Meldung fehl:
ORA-01779: Kann keine Spalte, die einer Basistabelle zugewiesen wird, verändern
Dies liegt daran, dass eine Klasse mehreren Studenten zugewiesen sein kann, und k_id somit kein Schlüssel vom view_all ist.

1.4.4 WITH CHECK OPTION

```
1 CREATE VIEW v1 AS
2 SELECT *
3 FROM studenten
4 WHERE s_klasse = 20 WITH CHECK OPTION;
```

Eine View v1 wird erstellt, die nur Studenten mit s_klasse 20 ausliest und die CHECK OPTION hat.

```
1 CREATE VIEW v2 AS
2 SELECT *
3 FROM v1
4 WHERE s_tel < 40000
```

Eine View v2 wird, auf Basis von v1, erstellt, die nur Studenten mit s_tel kleiner als 40000 ausliest.

```
1 INSERT INTO v2 VALUES (200, 'muster', 'peter', 40001, 2000, 20);
2 INSERT INTO v2 VALUES (201, 'muster', 'peter', 40001, 2000, 10);
```

Das erste Statement würde einen Eintrag erstellen der die Bedingungen von v1 erfüllt, die von v2 aber nicht. Das Statement wird erfolgreich durchgeführt, weil v2 keine CHECK OPTION hat. Das zweite Statement würde einen Eintrag erstellen der die Bedingungen von v1 und v2 nicht erfüllt, es bricht mit folgender Begründung ab:
ORA-01402: Verletzung der WHERE-Klausel einer View WITH CHECK OPTION
Dies zeigt, dass die CHECK OPTION von v1 für die Bedingung von v1 immer noch gilt, auch wenn das Statement auf v2 ausgeführt wird.

```
1 CREATE VIEW v3 AS
2 SELECT *
3 FROM v2
4 WHERE s_vname='peter' WITH CHECK OPTION;
```

Eine View v3 wird, auf Basis von v2, erstellt, die nur Studenten mit s_vname peter ausliest und die CHECK OPTION hat.

```
1 INSERT INTO v3 VALUES (202, 'muster', 'thomas', 30000, 2000, 20);
```

Dieses Statement verletzt nur die Bedingung von v3 und bricht mit folgender Meldung ab:
ORA-01402: Verletzung der WHERE-Klausel einer View WITH CHECK OPTION
Dies ist nachvollziehbar, da v3 die CHECK OPTION hat.

```
1 INSERT INTO v3 VALUES (202, 'muster', 'peter', 30000, 2000, 20);
```

Dieses Statement verletzt keine Bedingung und wird deshalb durchgeführt.

```
1 INSERT INTO v3 VALUES (203, 'muster', 'thomas', 30000, 2000, 10);
```

Dieses Statement verletzt nur die Bedingung von v1 und bricht mit folgender Meldung ab:
ORA-01402: Verletzung der WHERE-Klausel einer View WITH CHECK OPTION
Dies ist nicht überraschend, da schon gezeigt wurde, dass CHECK OPTION von verwendeten Views weiterhin gilt.

```
1 INSERT INTO v3 VALUES (204, 'muster', 'thomas', 40000, 2000, 20);
```

Dieses Statement verletzt nur die Bedingung von v2 und bricht mit folgender Meldung ab:
ORA-01402: Verletzung der WHERE-Klausel einer View WITH CHECK OPTION
Dies zeigt, dass die CHECK OPTION von v3 sich somit auf alle Bedingungen von verwendeten Views bezieht und egal ist, ob diese selber CHECK OPTION haben.

2 MS SQL Express 2012

2.1 Vorbereitung

Dieser Teil der Ausarbeitung wurde auf einem MS SQL Server Express 2012 auf einem Windows 7 Notebook durchgeführt. Um die User nicht auf dem Server und der Datenbank zu erstellen wurde die Datenbank dbarc03 als partial contained Datenbank erstellt, dafür musste dies zuerst aktiviert werden:

```
1 SP_CONFIGURE 'show advanced options',1;
2 RECONFIGURE;
3 GO;
4 SP_CONFIGURE 'contained database authentication',1;
5 RECONFIGURE;
6 GO;
7 SP_CONFIGURE 'show advanced options',0;
8 RECONFIGURE;
9 GO;
```

2.1.1 Einrichten User

Als sytem:

```
1 CREATE USER scott WITH PASSWORD = 'tiger';
2 CREATE USER nutzer01 WITH PASSWORD = 'nutzer01';
3 CREATE USER nutzer02 WITH PASSWORD = 'nutzer02';
4
5 GRANT CREATE ROLE TO scott;
6 GRANT CREATE VIEW TO scott;
7 GRANT CREATE TABLE TO scott;
8 GRANT CONTROL ON SCHEMA :: dbo TO scott;
```

Es werden die User scott, nutzer01 und nutzer02 analog zu der Durchführung in Oracle erstellt, CREATE SESSION Rechte sind in MS SQL nicht nötig um sich anzumelden. Scott erhält für das folgende Vorgehen Rechte um Rollen, Tabellen und Views zu erstellen, zusätzlich benötigt er noch Rechte um das Schema dbo zu verwalten.

2.2 Zugriffssteuerung mit User und Rollen

2.2.1 Tabellen erzeugen

Als scott:

```
CREATE TABLE klassen(
k_id INTEGER,
k_bezeichnung VARCHAR(20),
k_zimmer VARCHAR(10),
k_server VARCHAR(10));

CREATE TABLE studenten(
s_id INTEGER,
s_name VARCHAR(10),
s_vname VARCHAR(10),
s_tel VARCHAR(20),
s_konto_stand INTEGER,
s_klasse INTEGER);


INSERT INTO klassen values ( 10, 'ia00', '3333' , 'pluto');
INSERT INTO klassen values ( 20, 'ia01', '2222' , 'saturn');

INSERT INTO studenten values( 101, 'meier', 'hans', '11111', 5000, 10);
INSERT INTO studenten values( 102, 'hirt', 'otto', '22222', -100, 10);
INSERT INTO studenten values( 103, 'kok', 'thomas', '33333', 1000, 20);
INSERT INTO studenten values( 104, 'guzman', 'anna', '44444', 3000, 20);
INSERT INTO studenten values( 105, 'lorch', 'felix', '45678', 7000, 20);
```

Die Tabellen werden analog zur Oracle Übung erstellt, der Datentyp NUMBER(9) wurde durch INTEGER ersetzt und VARCHAR2 durch VARCHAR, da diese Typen unter MS SQL nicht existieren.

2.2.2 View erstellen

Als scott:

```
CREATE VIEW view1 AS
select s_name, s_vname, k_zimmer, k_server
FROM studenten, klassen
WHERE s_klasse = k_id
```

Scott erstellt eine View view1 in der Daten aus studenten und klassen angezeigt werden.

2.2.3 Rollen definieren

Als scott:

```
CREATE ROLE view1_verwalter;
CREATE ROLE view1_nutzer;
```

Scott erstellt die Rollen view1_verwalter und view1_nutzer.

2.2.4 Den Rollen Rechte zuweisen

Als scott:

```
1 GRANT INSERT,SELECT,UPDATE,DELETE ON studenten TO view1_verwalter;
2 GRANT INSERT,SELECT,UPDATE,DELETE ON klassen TO view1_verwalter;
```

Scott erteilt der Rolle view1_verwalter die Rechte Daten in studenten und klassen zu sehen und zu verändern.

Als scott:

```
1 GRANT SELECT ON view1 TO view1_nutzer;
```

Scott erteilt der Rolle view1_nutzer das Recht die Daten in view1 zu lesen.

2.2.5 Den User Rollen zuweisen

Als scott:

```
1 GRANT view1_verwalter TO nutzer01;
2 GRANT view1_nutzer TO nutzer02;
```

Scott weist nutzer01 die Rolle view1_verwalter und nutzer02 die Rolle view2_nutzer zu.

2.2.6 Überprüfung der Rechte der beiden Rollen

Die folgenden Statements werden zur Überprüfung der Rechte verwendet:

```
1  SELECT * FROM scott.studenten;
2  INSERT INTO scott.studenten VALUES (106, 'peter', 'muster', 12345,2000,20);
3  UPDATE scott.studenten SET s_tel = 54321 WHERE s_id = 106;
4  DELETE FROM scott.studenten WHERE s_id = 106;
5
6  SELECT * FROM scott.klassen;
7  INSERT INTO scott.klassen VALUES (30, 'ia02', 1111, 'uranus');
8  UPDATE scott.klassen SET k_zimmer = 4444 WHERE k_id = 30;
9  DELETE FROM scott.klassen WHERE k_id = 30;
10
11 SELECT * FROM scott.view1;
12 INSERT INTO scott.view1 VALUES ('peter','muster',2222,'saturn');
13 UPDATE scott.view1 SET s_name = 'kook' WHERE s_vname = 'thomas';
14 DELETE FROM scott.view1 WHERE s_vname = 'thomas';
```

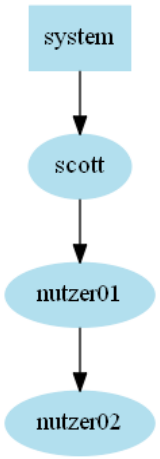
Was kann der Verwalter lesen und bearbeiten?

Die Statements in den Linien 1 bis 9 können von nutzer01 ausgeführt werden. Wenn nutzer01 das Statement in Linie 11 ausführen will erhält er folgende Fehlermeldung:
The SELECT permission was denied on the object 'view1', database 'dbarc03', schema 'dbo'.
Im Gegensatz zu Oracle bestätigt MS SQL somit, das die View existiert. Bei den Statements in Linie 12 bis 14 erhält nutzer01 folgende Meldung:
View or function 'view1' is not updatable because the modification affects multiple base tables.
MS SQL wertet hier also die Gültigkeit des Statements vor den Berechtigungen aus, was dazu führt das nutzer01 Informationen über gültige Statements auf view1 erhält, obwohl er keinen Zugriff darauf hat, abgesehen von den Meldungen funktioniert das Berechtigungskonzept hier analog zu Oracle.

Was kann der Nutzer lesen und bearbeiten?

Wenn nutzer02 die Statements in den Linien 1 bis 9 durchführen will erhält er folgende Meldungen:
The SELECT permission was denied on the object 'klassen', database 'dbarc03', schema 'dbo'.
The SELECT permission was denied on the object 'studenten', database 'dbarc03', schema 'dbo'.
Auch hier zeigt MS SQL das die Tabellen zwar vorhanden sind, nutzer02 aber keine Berechtigungen hat. Nutzer02 kann das Statement in Linie 11 ausführen, bei den Statements in Linien 12 bis 14 erhält er auch folgende Meldung:
View or function 'view1' is not updatable because the modification affects multiple base tables.
Auch hier wird die Gültigkeit vor den Berechtigungen ausgewertet, die Berechtigungen funktionieren aber auch analog zu Oracle.

2.3 Zugriffsrechte: Objekt- und Systemrechte



2.3.1 Objektrechte

Als scott:

```
1 CREATE VIEW view2 AS
2 select s_name, s_vname, k_zimmer, k_server
3 FROM studenten, klassen
4 WHERE s_klasse = k_id;
5
6 GRANT SELECT ON view2 TO nutzer01 WITH GRANT OPTION;
```

Scott erstellt eine View view2 und erteilt nutzer01 das SELECT-Recht dazu, mit der Option dieses weiterzugeben.

Als nutzer01:

```
1 GRANT SELECT ON view2 TO nutzer02;
```

Nutzer01 gibt das SELECT-Recht an nutzer02 weiter.

Als nutzer02:

```
1 SELECT * FROM scott view2;
```

Nutzer02 kann die Daten aus view2 erfolgreich weitergeben.

Als scott:

```
1 REVOKE SELECT ON view2 FROM nutzer01 CASCADE;
```

Scott entzieht nutzer01 die Rechte auf view2 wieder, das Schlüsselwort CASCADE muss bei MS SQL angegeben werden, sobald Rechte mit WITH GRANT OPTION erteilt wurden, auch wenn diese noch nicht weiter gegeben wurden, ansonsten bricht das REVOKE-Statement mit folgender Meldung ab:
To revoke or deny grantable privileges, specify the CASCADE option.

Als nutzer02:

```
1 SELECT * FROM scott view2;
```

Das Statement schlägt mit folgender Meldung fehl:
The SELECT permission was denied on the object 'view2', database 'dbarc03', schema 'dbo'.
Die Berechtigungen wurden also, wie das Schlüsselwort CASCADE impliziert, kaskadierend entzogen, MS SQL schützt also davor, Rechte unbewusst kaskadierend zu entziehen.

2.3.2 Systemrechte

Als system:

```
1 GRANT CREATE VIEW TO nutzer01 WITH GRANT OPTION;
```

Nutzer01 werden die Rechte erteilt, eine View zu erstellen mit der Option dieses Recht weiterzugeben. Da MS SQL das WITH ADMIN OPTION nicht kennt muss hier auch WITH GRANT OPTION verwendet werden.

als nutzer01:

```
1 GRANT CREATE VIEW TO nutzer02;
```

Nutzer01 gibt das Recht an nutzer02 weiter.

Als system:

```
1 REVOKE CREATE VIEW FROM nutzer01 CASCADE;
```

Da das Recht wieder mit GRANT OPTION weitergegeben wurde, muss auch hier beim entfernen das Schlüsselwort CASCADE angegeben werden, und Systemrechte werden dadurch, im Gegensatz zu Oracle, kaskadierend entzogen.

2.4 Views

2.4.1 Rechte von Views

Als system:

```
1 GRANT CREATE VIEW TO nutzer01;
2 GRANT ALTER ON SCHEMA :: dbo TO nutzer01;
```

Nutzer01 wird das Recht Views zu erstellen erteilt, gleichzeitig benötigt er auch das Recht, das Schema dbo zu ändern.

Als nutzer01:

```
1 CREATE VIEW cheat_view AS
2 SELECT * FROM studenten;
```

Nutzer01 kann die View nun erstellen, was unter Oracle nicht ging, dies liegt an der ALTER Berechtigung auf dbo. Als nutzer01:

```
1 SELECT * FROM cheat_view
```

Nutzer01 versucht nun die Daten der eben erstellten view anzuzeigen, dies schlägt mit folgender Meldung fehl:
The SELECT permission was denied on the object 'cheat_view', database 'dbarc03', schema 'dbo'.
Dies zeigt, dass nutzer01 keine Berechtigungen hat Views von dbo anzuzeigen, auf die er nicht explizit Berechtigungen hat.

Als nutzer01:

```
1 DROP VIEW cheat_view;
```

Nutzer01 löscht die View wieder.

Als system:

```
1 CREATE SCHEMA nutzer01 AUTHORIZATION nutzer01;
```

Nutzer01 erhält sein eigenes Schema, auf dem er volle Berechtigungen hat.

Als nutzer01:

```
1 CREATE VIEW nutzer01.cheat_view AS
2 SELECT * FROM dbo.studenten;
```

Nutzer01 erstellt die View im nutzer01 Schema.

Als nutzer01:

```
1 SELECT * FROM nutzer01.cheat_view;
2 GRANT SELECT ON nutzer01.cheat_view TO nutzer02;
```

Nutzer01 kann die View nun selber erfolgreich anzeigen, er gibt die SELECT-Rechte an nutzer02 weiter, welcher die Daten ja nicht sehen dürfte.

Als nutzer02:

```
1 SELECT * FROM nutzer01.cheat_view
```

Nutzer02 versucht nun die Daten auszulesen, was mit folgender Meldung abbricht:
The SELECT permission was denied on the object 'studenten', database 'dbarc03', schema 'dbo'.
Diese Meldung zeigt eindeutig, das die Rechte auf dbo fehlen, und nicht auf nutzer01, auch bei MS SQL können Views also nicht verwendet werden, um Berechtigungen unerlaubt weiterzugeben.

2.4.2 DDL-Änderungen an den Basistabellen

Alle Statements in folgendem Absatz werden als scott durchgeführt.

```
1 ALTER TABLE studenten ADD birthday TIMESTAMP;
```

Scott fügt eine Spalte brithday zu studenten hinzu.

```
1 SELECT * FROM view1;
```

View1 lässt sich immer noch unverändert abrufen, da birhday von ihr nicht abgefragt wird.

```
1 ALTER TABLE studenten DROP COLUMN birthday;
```

Scott entfernt birthday wieder.

```
1 ALTER TABLE studenten DROP COLUMN s_name;
```

Scott entfernt die Spalte s_name.

```
1 SELECT * FROM view1;
```

View1 kann nun nicht mehr angezeigt werden, sondern bricht mit folgender Meldung ab:
Invalid column name 's_name'. Could not use view or function 'view1' because of binding errors.
Dies zeigt, das auch unter MS SQL nur Spalten, welche auch verwendet werden, die View beeinflussen.

```
1 ALTER TABLE studenten ADD s_name VARCHAR(10);
```

Scott fügt die Spale wieder ein, view1 kann wieder angezeigt werden, enthält jetzt aber NULL in s_name.

2.4.3 Updatable Views

Alle Statements in folgendem Absatz werden als scott durchgeführt.

```
1 CREATE VIEW view_distinct AS
2 SELECT DISTINCT s_vname
3 FROM studenten;
4
5 UPDATE view_distinct SET s_vname='ana' WHERE s_vname='anna';
```

Scott erstellt eine View, welche das Schlüsselwort DISTINCE verwendet, und versucht ein UPDATE-Statement auzuführen, was abbricht:
Cannot update the view or function 'view_distinct' because it contains aggregates, or a DISTINCT or GROUP BY clause, or PIVOT or UNPIVOT operator.
Die Meldung beschreibt sehr ausführlich, welche Schlüsselwörter dazu führen, dass über eine View keine Daten verändert werden können.

```
1 CREATE VIEW view_all AS
2 SELECT *
3 FROM studenten
4 INNER JOIN klassen ON s_klasse=k_id;
```

Scott erstellt nun eine View view_all die alle Daten der Studenten und ihren Klassen anzeigt.

```
1 UPDATE view_all SET s_name='muster' WHERE s_id=101;
```

Scott versucht nun den Namen des Studenten 101 über die View zu ändern, dies ist im Gegensatz zu Oracle möglich, obwohl keine Schlüssel definiert sind.

```
1 UPDATE view_all SET k_zimmer=4444 WHERE k_id=20;
```

Scott versucht nun das Zimmer der Klasse 20 über die View zu ändern, auch dies funktioniert, obwohl die Klasse 20 mehrmals vorkommt, MS SQL kennt folglich keinen Schutz wie Oracle.

2.4.4 WITH CHECK OPTION

```
1 CREATE VIEW v1 AS
2 SELECT *
3 FROM studenten
4 WHERE s_klasse = 20 WITH CHECK OPTION;
```

Scott erstellt eine View v1 die alle Studenten aus Klasse 20 anzeigt, und definiert die CHECK OPTION.

```
1 CREATE VIEW v2 AS
2 SELECT *
3 FROM v1
4 WHERE s_tel < 40000
```

Scott erstellt eine View v2, basierend auf v2, die nur Einträge mit s_tel kleiner als 40000 anzeigt.

```
1 INSERT INTO v2 VALUES (200, 'muster', 'peter', 40001, 2000, 20);
2 INSERT INTO v2 VALUES (201, 'muster', 'peter', 40001, 2000, 10);
```

Das erste Statement wird ausgeführt, da es nur die Bedingung der View ohne CHECK OPTION verletzt. Das zweite Statement bricht mit folgender Begründung ab:
The attempted insert or update failed because the target view either specifies WITH CHECK OPTION or spans a view that specifies WITH CHECK OPTION and one or more rows resulting from the operation did not qualify under the CHECK OPTION constraint.
Die CHECK OPTION betrifft also wie in Oracle die Bedingung von v2 nicht.

```
1 CREATE VIEW v3 AS
2 SELECT *
3 FROM v2
4 WHERE s_vname='peter' WITH CHECK OPTION;
```

Scott erstellt eine View v3, basierend auf v2, mit der Bedingung das der Vorname peter ist und der CHECK OPTION.

```
1 INSERT INTO v3 VALUES (202, 'muster', 'thomas', 30000, 2000, 20);
```

Da dieses Statement die Bedingung von v3 verletzt, bricht es mit folgender Meldung ab:
The attempted insert or update failed because the target view either specifies WITH CHECK OPTION or spans a view that specifies WITH CHECK OPTION and one or more rows resulting from the operation did not qualify under the CHECK OPTION constraint.

```
1 INSERT INTO v3 VALUES (202, 'muster', 'peter', 30000, 2000, 20);
```

Dieses Statement verletzt keine Bedingungen und läuft durch.

```
1 INSERT INTO v3 VALUES (203, 'muster', 'peter', 30000, 2000, 10);
```

Da dieses Statement die Bedingung von v1 verletzt, bricht es mit folgender Meldung ab:
The attempted insert or update failed because the target view either specifies WITH CHECK OPTION or spans a view that specifies WITH CHECK OPTION and one or more rows resulting from the operation did not qualify under the CHECK OPTION constraint.

```
1 INSERT INTO v3 VALUES (204, 'muster', 'peter', 40000, 2000, 20);
```

Da dieses Statement die Bedingung von v2 verletzt, bricht es mit folgender Meldung ab:
The attempted insert or update failed because the target view either specifies WITH CHECK OPTION or spans a view that specifies WITH CHECK OPTION and one or more rows resulting from the operation did not qualify under the CHECK OPTION constraint.
Die CHECK OPTION hat also wie bei Oracle auch unter MS SQL den Effekt, das Bedingungen von verwendeten Views auch geprüft werden, wenn nur die verwendende View die CHECK OPTION gesetzt hat.

3 MySQL

Für diese Ausarbeitung verwenden wir XAMPP 1.7.3 mit MySQL 5.5.27.

3.1 Vorbereitung

3.1.1 Einrichtung User

```
1 CREATE USER scott@localhost IDENTIFIED BY 'tiger';
2 CREATE USER nutzer01@localhost IDENTIFIED BY 'nutzer01';
3 CREATE USER nutzer02@localhost IDENTIFIED BY 'nutzer02';
4 GRANT ALL PRIVILEGES ON `dbarc03` . * TO 'scott'@'localhost';
```

Bei MySQL muss bei der Erstellung von Benutzern neben dem Namen und der Identifizierungsmethode auch definiert werden vom welchem Server aus sich der Nutzereinloggen darf. Da unser MySQL Server lokal läuft, wird hier 'localhost' angegeben. Daneben gibt es auch IP und Servername, sowie die Berechtigung ohne Einschränkungen (%) und die Regelung der Zugriffe per Hosttabelle.

3.1.2 Einrichtung Datenbank

```
1 CREATE DATABASE dbarc03;
```

3.2 Zugriffssteuerung mit User und Rollen

3.2.1 Tabellen erzeugen

Als Benutzer 'scott':

```
1  CREATE TABLE studenten(
2  s_id int(9) PRIMARY KEY,
3  s_name VARCHAR(10),
4  s_vname VARCHAR(10),
5  s_tel VARCHAR(20),
6  s_konto_stand int(9),
7  s_klasse int(9));
8
9  CREATE TABLE klassen(
10 k_id int(9),
11 k_bezeichnung VARCHAR(20),
12 k_zimmer VARCHAR(10),
13 k_server VARCHAR(10));
14
15 INSERT INTO klassen values( 10, 'ia00', '3333' , 'pluto');
16 INSERT INTO klassen values( 20, 'ia01', '2222' , 'saturn');
17
18 INSERT INTO studenten values( 101, 'meier', 'hans', '11111', 5000, 10);
19 INSERT INTO studenten values( 102, 'hirt', 'otto', '22222', -100, 10);
20 INSERT INTO studenten values( 103, 'kok', 'thomas', '33333', 1000, 20);
21 INSERT INTO studenten values( 104, 'guzman', 'anna', '44444', 3000, 20);
```

```
22 INSERT INTO studenten values( 105, 'lorch', 'felix', '45678', 7000, 20);
```

3.2.2 View erstellen

```
1 CREATE VIEW view1 AS
2 SELECT s_name, s_vname, k_zimmer, k_server
3 FROM studenten, klassen
4 WHERE s_klasse = k_id
```

3.2.3 Rechte an User verteilen

Als 'root':

```
1 REVOKE ALL PRIVILEGES ON `dbarc03` . * FROM scott@localhost;
2 GRANT SELECT, INSERT, UPDATE, DELETE, CREATE, DROP, INDEX, ALTER, CREATE
3 VIEW, SHOW VIEW
4 ON `dbarc03` . *
5 TO scott@localhost WITH GRANT OPTION;
```

Scott kann nun seine Rechte frei weitergeben.

Als Benutzer 'scott':

```
1 GRANT SELECT, INSERT, UPDATE, DELETE ON studenten TO
2 nutzer01@localhost;
3
4 GRANT SELECT, INSERT, UPDATE, DELETE ON klassen TO
5 nutzer01@localhost;
6
7 GRANT SELECT ON view1 TO nutzer02@localhost;
```

In MySQL gibt es, im Gegensatz zu Oracle, keine Rollen. Man muss also jedem Nutzer die Rechte direkt zuteilen.
Deswegen gibt hier 'scott' dem Benutzer 'nutzer01' select-, insert-, update- und delete-Rechte für die beiden Tabellen 'studenten' und 'klassen' und dem Benutzer 'nutzer02' select-Rechte an der View 'view1'.

3.2.4 Überprüfung der Rechte der beiden Nutzer

Statement 1

```
1 SELECT * FROM dbarc03.studenten;
2 INSERT INTO dbarc03.studenten VALUES (106, 'peter', 'muster', 12345,2000,20);
3 UPDATE dbarc03.studenten SET s_tel = 54321 WHERE s_id = 106;
4 DELETE FROM dbarc03.studenten WHERE s_id = 106;
```

Statement 2

```
1 SELECT * FROM dbarc03.klassen;
2 INSERT INTO dbarc03.klassen VALUES (30, 'ia02', 1111, 'uranus');
3 UPDATE dbarc03.klassen SET k_zimmer = 4444 WHERE k_id = 30;
4 DELETE FROM dbarc03.klassen WHERE k_id = 30;
```

Statement 3

```
1 SELECT * FROM dbarc03.view1;
2 INSERT INTO dbarc03.view1 VALUES ('peter','muster',2222,'saturn');
3 UPDATE dbarc03.view1 SET s_name = 'kook' WHERE s_vname = 'thomas';
4 DELETE FROM dbarc03.view1 WHERE s_vname = 'thomas';
```

Die obigen 3 Statements werden jeweils von 'nutzer01' und 'nutzer02' ausgeführt.

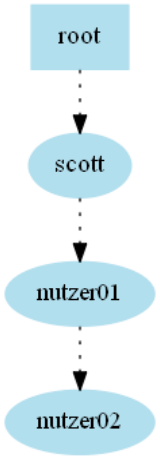
Was kann nutzer01 lesen und bearbeiten?

'nutzer01' kann die ersten beiden Statements ausführen. Er kann also Daten der beiden Tabellen 'studenten' und 'klassen' auslesen, ergänzen, verändern und löschen. Er kann keinerlei Operationen auf der View 'view1' ausführen. Dies verhält sich also wie bei Oracle.

Was kann nutzer02 lesen und bearbeiten?

Im Gegensatz zu 'nutzer01' kann 'nutzer02' keinerlei Operationen an den Tabellen 'studenten' und 'klassen' durchführen. Wie erwartet kann er in Statement 3 die Daten aus der View 'view1' lediglich auslesen. Auch hier verhält sich MySQL genauso wie bei Oracle.

3.3 Zugriffsrechte: Objekt- und Systemrechte



Berechtigungen unter MySQL
Berechtigungen werden unter MySQL in 4 Stufen klassifiziert:

Globale Ebene
Globale Rechte gelten für alle Datenbanken auf einem bestimmten Server und werden in der Systemtabelle mysql.user gespeichert
Datenbankebene
Datenbankrechte gelten für alle Tabellen einer bestimmten Datenbank und werden in der Systemtabelle mysql.db gespeichert
Tabellenebene
Tabellenrechte gelten für alle Spalten einer bestimmten Tabelle und werden in der Systemtabelle mysql.tables_priv gespeichert
Spaltenebene
Spaltenrechte gelten für einzelne Spalten einer bestimmtem Tabelle und werden in der Systembtabelle mysql.columns_priv gespeichert

Quelle: 'SQL in a Nutshell' by Kevin E. Klinke, 2.Auflage, S: 307

3.3.1 Objektrechte

Um die Objektrechte überprüfen zu können erstellen wir als Benutzer 'scott' eine zweite View namens 'view2'. Wir geben 'nutzer01' select-Rechte für diese View.

```
1 CREATE VIEW view2 AS
2 select s_name, s_vname, k_zimmer, k_server
3 FROM studenten, klassen
4 WHERE s_klasse = k_id;
5
6 GRANT SELECT ON dbarc03.view2 TO nutzer01@localhost WITH GRANT OPTION;
```

'nutzer01' gibt diese select-Rechte an der View 'view2' nun weiter an 'nutzer02'.

```
1 GRANT SELECT ON dbarc03.view2 TO nutzer02@localhost;
```

'nutzer02' testet, ob er nun das Recht hat Daten aus 'view2' auszulesen.

```
1 SELECT * FROM dbarc03.view2;
```

Erfolgreich.

Nun nimmt 'scott' die selec-Rechte an der View 'view2' dem Nutzer 'nutzer01' wieder weg.

```
1 REVOKE SELECT ON dbarc03.view2 FROM nutzer01@localhost;
```

Anschliessend testen wir als 'nutzer02' ob wir immernoch select-Rechte auf 'view2' haben oder ob diese mit dem Entfernen der Rechte von 'nutzer01' ebenfalls erloschen sind.

```
1 SELECT * FROM dbarc03.view2;
```

Immernoch erfolgreich.

MySQL verhält sich hier grundlegend anders als Oracle. Es gibt zwei Arten einem Nutzer Rechte zu geben. Mit 'WITH GRANT OPTION' oder ohne.
Ohne 'WITH GRANT OPTION' können die Rechte nicht weitergegeben werden. Mit 'WITH GRANT OPTION' können Rechte uneingeschränkt weitergegeben werden. Anders als bei Oracle entstehen so keine Abhängigkeiten und somit keine Kaskadeneffekte.

3.3.2 Systemrechte

Um die Systemrechte zu überprüfen erhält Nutzer 'scott' das Recht neue Nutzer zu erstellen.

```
1 GRANT CREATE USER ON *.* TO scott@localhost WITH GRANT OPTION;
```

'scott' gibt dieses Recht anschliessen an 'nutzer01' weiter.

```
1 GRANT CREATE USER ON *.* TO nutzer01@localhost;
```

Nun entzieht der Root 'scott' das Recht neue Nutzer zu erstellen wieder.

```
1 REVOKE CREATE USER ON *.* FROM scott@localhost;
```

'nutzer01' testet ob er immernoch neue Nutzer erstellen kann.

```
1 CREATE USER test@localhost IDENTIFIED BY 'test';
```

Erfolgreich.

Auch hier entsteht keine Kaskade. In MySQL verhalten sich Objekt- und Systemrechte identisch.

3.4 Views

3.4.1 Rechte von Views

Zuerst erhält 'nutzer01' das Recht neue Views zu erstellen.

```
1 GRANT CREATE VIEW ON `dbarc03` .* TO nutzer01@localhost;
```

'nutzer01' erstellt hierauf die View 'cheat_view' auf die Daten in der Tabelle 'studenten'.

```
1 CREATE VIEW cheat_view AS
2 SELECT * FROM dbarc03.studenten;
```

Erfolgreich.

'nutzer01' gibt nun 'nutzer02' das select-Recht für die View 'cheat_view'.

```
1 GRANT SELECT ON cheat_view TO nutzer02@localhost;
```

Error Code: 1142. SELECT command denied to user 'nutzer01'@'localhost' for table 'cheat_view'

In MySQL wird die View, welche 'nutzer01' erstellt hat, nicht in einem Tablespace gespeichert, welcher 'nutzer01' gehört. Folglich hat er keinerlei Rechte auf der von ihm erstellten View und kann diese so natürlich auch nicht weitergeben.

3.4.2 DDL-Änderungen an den Basistabellen

Erst überprüfen wir wie sich die View 'view1' verhält, wenn der Nutzer 'scott' die Tabelle 'studenten' um eine Spalte 'birthday' erweitert.

```
1 ALTER TABLE studenten ADD birthday TIMESTAMP;
```

```
1 SELECT * FROM view1;
```

Es entstehen keinerlei Probleme, da 'view1' durch die neu Spalte nicht beeinträchtigt wird.

Als nächstes löscht 'scott' die soeben erstellte Spalte 'birthday' wieder aus der Tabelle 'studenten', zusammen mit der vorhandenen Spalte 's_name'.

```
1 ALTER TABLE studenten DROP COLUMN birthday;
2 ALTER TABLE studenten DROP COLUMN s_name;
```

```
1 SELECT * FROM view1;
```

Error Code: 1356. View 'dbarc03.view1' references invalid table(s) or column(s) or function(s) or definer/invoker of view lack rights to use them

Wie man aus der Fehlermeldung erkennen kann, referenziert 'view1' immernoch auf die soeben gelöschte Spalte 's_name'.

Wir fügen die Spalte 's_name' wieder hinzu um so die View 'view1' wieder zu 'reparieren'.

```
1 ALTER TABLE studenten ADD s_name VARCHAR(10);
```

3.4.3 Updatable Views

Nun wollen wir überprüfen was passiert, wenn man versucht über eine View die Daten der Tabelle zu verändern.

Nutzer 'scott' erstellt hierfür eine neue View 'view_distinct'. Anschliessend versucht er die Daten der Spalte 's_vname' via update-Statement zu verändern.

```
1 CREATE VIEW view_distinct AS
2 SELECT DISTINCT s_vname
3 FROM studenten;
4
5 UPDATE view_distinct SET s_vname='ana' WHERE s_vname='anna';
```

Error Code: 1288. The target table view_distinct of the UPDATE is not updatable

MySQL verhält sich genauso wie Oracle.

Anschliessend erstellt 'scott' eine weitere View 'view_all', diesmal ohne distinct.

```
1 CREATE VIEW view_all AS
2 SELECT *
3 FROM studenten
4 INNER JOIN klassen ON s_klasse=k_id;
```

'scott' versucht nun die Daten der Tabelle über ein update-Statement auf 'view_all' zu verändern.

```
SELECT * FROM view_all;

UPDATE view_all SET s_name='muster' WHERE s_id=101;
```

Dies gelingt, wie bei Oracle.

```
UPDATE view_all SET k_bezeichnung='test' WHERE k_id=10;
```

3.4.4 WITH CHECK OPTION

Die **WITH CHECK OPTION**-Klausel kann für eine veränderbare View benutzt werden, um Einfügungen oder Änderungen an Zeilen zu verhindern, welche die **WHERE**-Klausel des SELECT-Statements nicht erfüllen. In einer **WITH CHECK OPTION**-Klausel einer veränderbaren View legen die Schlüsselwörter **LOCAL** und **CASCADED** den Rahmen der Überprüfungen fest, wenn die View auf Grundlage einer anderen View definiert wurde. Das Schlüsselwort **LOCAL** schränkt die **CHECK OPTION** auf die View ein, die gerade definiert wird, während **CASCADED** dafür sorgt, dass sich die Prüfungen auch auf die zugrunde liegenden Views erstrecken. Wenn keines der beiden Schlüsselwörter angegeben ist, ist **CASCADED** der Standard.

Quelle: http://dev.mysql.com/doc/refman/5.1/de/create-view.html

Nun wollen wir das Verhalten von **WITH CHECK OPTION** in MySQL untersuchen.

```
CREATE VIEW v1 AS
SELECT *
FROM studenten
WHERE s_klasse = 20 WITH CHECK OPTION;
```

```
CREATE VIEW v2 AS
SELECT *
FROM v1
WHERE s_tel < 40000
```

```
INSERT INTO v2 VALUES (200, 'muster', 'peter', 40001, 2000, 20);
INSERT INTO v2 VALUES (201, 'muster', 'peter', 40001, 2000, 10);
```

Erfolgreich.

```
CREATE VIEW v3 AS
SELECT *
FROM v2
WHERE s_vname='peter' WITH CHECK OPTION;
```

```
INSERT INTO v3 VALUES (202, 'muster', 'thomas', 30000, 2000, 20);
```

Error Code: 1369. CHECK OPTION failed 'dbarc03.v3'

```
INSERT INTO v3 VALUES (202, 'muster', 'peter', 30000, 2000, 20);
```

Error Code: 1369. CHECK OPTION failed 'dbarc03.v3'

```
1 INSERT INTO v3 VALUES (203, 'muster', 'thomas', 30000, 2000, 10);
```

Error Code: 1369. CHECK OPTION failed 'dbarc03.v3'

```
1 INSERT INTO v3 VALUES (204, 'muster', 'thomas', 40000, 2000, 20);
```

Error Code: 1369. CHECK OPTION failed 'dbarc03.v3'

Wenn man unter MySQL mit aufeinander aufbauenden Views arbeiten will, muss man darauf achten, dass wenn man eine View mit **WITH CHECK OPTION** erstellt, dass die darüber liegenden Views ebenfalls **WITH CHECK OPTION** trägt, ansonten kann es vorkommen, dass die Datenbank nicht wie erwartet reagiert.

WITH CASCADED CHECK OPTION verhält sich wie zu erwarten war.

Wenn nach einer View mit **WITH CHECK OPTION** eine View ohne folgt, kann diese die **CHECK OPTION** der vorherigen View aushebeln. Ohne **WITH CHECK OPTION** ist die Einstellung der Basistabelle relevant.

3.5 Standard SQL

Wir wollen nun sehen was der SQL Standard 1999 eigentlich vorsehen würde und ob Oracle, MySQL und MS SQL davon abweichen oder nicht.

3.5.1 Zugriffssteuerng mit Benutzern und Rollen

Hier weicht MySQL vom Standard ab, denn der SQL Standard sieht bei der Benutzersteuerung ein Rollenkonzept vor wie es sowohl Oracle, als auch MS SQL implementiert haben.

3.5.2 Zugriffsrechte: Objekt- und Systemrechte

WITH GRAN OPTION

Seit dem SQL Standard von 1992 ist **WITH GRANT OPTION** ein fester Bestandteil des Standards. Alle drei Datenbank-Systeme unterstützen es.

WITH ADMIN OPTION

Gehört seit 1999 zum SQL Standard. Im Gegensatz zu **WITH GRANT OPTION** wird es allerdings nur von Oracle unterstützt.

REVOKE

Seit 1992 ist **REVOKE** Bestandteil des SQL Standards.

REVOKE soll laut Standard die beiden Optionen **RESTRICT** und **CASCADE** unterstützen. **RESTRICT** scheint allerdings von keinem der drei Datenbank-Systeme unterstützt zu werden. Auch was **CASCADE** angeht, weichen alle Systeme vom Standard ab.

Bei MS SQL muss bei einem **REVOKE** die Option **CASCADE** angegeben werden, falls die Rechte mit **WITH GRANT OPTION** gegeben wurden.

Oracle entfernt Rechte standardmässig kaskadierend wenn sie mit **WITH GRANT OPTION** vergeben wurden. Bei **WITH ADMIN OPTION** werden sie allerdings nicht kaskadierend entfernt.

MySQL stellt keine Möglichkeit bereit um Rechte kaskadierend entfernen zu können.

3.5.3 Views

Rechte auf Views

Laut SQL Standard von 1999 kann ein Benutzer keine höheren Rechte auf einer View erhalten als er sie auf der Basistabelle hat. Ausnahmsweise verhandeln sich alle drei Datenbank-Systeme gemäss des Standards.

Updateable Views

Seit 1992 gibt der SQL Standard vor wann View wie veränderbar sind. Alle Datenbank-Systeme verhalten sich diesem Standard entsprechend. Bei Oracle kann eine Zeile der View nur verändert werden, wenn die zu verändernde Basistabelle **key-preserved** ist.

WITH CHECK OPTION

Im SQL Standard von 1992 wurde der Parameter **WITH CHECK OPTION** das erste Mal definiert. Er wird von allen drei Systemen der Definition gemäss umgesetzt. Neben **WITH CHECK OPTION** gibt es im Standard noch zwei weitere Parameter namens **LOCAL** und **CASCADED**. Diese werden allerdings nur von MySQL unterstützt.

4 Reflexion

Es ist erstaunlich zu sehen, dass es seit über 20 Jahren einen gültigen SQL Standard gibt und dieser von keinem der drei getesteten Datenbank-Systemen vollkommen berücksichtigt wird. Jedes System interpretiert Dinge aus dem Standard anders oder lässt sie komplett aussen vor. (z.B. Fehlende Rollen bei MySQL)
Dies führt zu der Erkenntnis, dass man sich nicht einfach auf den SQL Standard berufen kann, sondern sich je nach System selbst grundlegende Dinge wie unterstützte Datentypen unterscheiden können, dies ist auch für Programmierer relevant, die das System zwar nur verwenden und nicht administrieren, aber doch wissen müssen wie sich die das System speziell verhält.