1. Einleitung

Dieses Dokument ist im Rahmen des Moduls dbarc entstanden und geht auf die Fragenstellungen der Übung 3 ein. Konkret befasst sich diese mit der Zugriffssteuerung und den Views. Für eine bessere Übersicht sind SQL-Befehle in Konsolenschrift und die Syntax zusätzlich in Grossbuchstaben beschrieben, ggf. werden diese mit weiterführenden Befehlen ergänzt.

2. Vorbereitung (als user system)

```
-- Create two other users
CREATE USER charlie IDENTIFIED BY charliesSuperSecurePassword;
CREATE USER snoopy IDENTIFIED BY snoopysSuperSecurePassword;

-- Grant essential privileges
GRANT CREATE SESSION TO charlie;
GRANT CREATE SESSION TO snoopy;

-- Allow user scott to create rules
GRANT CREATE ROLE TO scott;

-- Allow user scott to create views
GRANT CREATE VIEW TO scott;
```

3. Views

3.1 DDL-Änderungen an den Basistabellen (als user scott)

Zunächst wird eine View erstellt:

```
CREATE OR REPLACE VIEW emp_seniority_view
AS SELECT ename, hiredate
FROM emp;
```

Zusätzliche Spalte in die Tabelle `EMP` einfügen:

```
ALTER TABLE emp ADD FIRED date;
```

Diese Operation konnte ohne zusätzliche Privilages mit dem User Scott ausgeführt werden. Dies liegt daran das er der Owner der Tabelle `emp` ist: Wir verifizieren dies mit:

```
-- List tables owned by user scott
SELECT * FROM  CAT;
```

	TABLE_NAME	TABLE_TYPE
1	DEPT	TABLE
2	EMP	TABLE
3	BONUS	TABLE
4	SALGRADE	TABLE
5	EMP_SENIORITY_VIEW	VIEW

3.2.1 Updateable Views

Grundsätzlich sind `UPDATE`, `INSERT` und `DELETE` Befehle auf Views problemlos möglich. Allerdings mit folgenden Einschränkungen:

Einschränkung	DELETE	UDPATE	INSERT
Gruppenfunktionen	x	x	x
`GROUP BY` Klausel	x	x	x
das Schlüsselwort `DISTINCT`	x	x	x

Einschränkung	DELETE	UDPATE	INSERT
die Pseudospalte ROWNUM	x	x	x
durch Ausdrücke definierte Spalten		x	x
NOT NULL Spalten der Basistabelle, die nicht durch die View ausgewählt werden			x

Funktionierendes Beispiel an einem UPDATE:

```
UPDATE EMP_SENIORITY_VIEW SET HIREDATE = CURRENT_TIMESTAMP WHERE ENAME = 'SMITH';
```

	ENAME	HIREDATE
1	SMITH	2020-03-09 09:24:34
2	ALLEN	1981-02-20 00:00:00
3	WARD	1981-02-22 00:00:00

Nun erstellen wir eine neue View welche eine Funktion enthält:

```
-- Create view
CREATE OR REPLACE VIEW emp_function_demo_view
    (ENAME,SAL)
AS SELECT concat(ENAME, JOB), SAL
   FROM emp;

-- show result
SELECT * FROM emp_function_demo_view;
```

	ENAME	SAL
1	SMITHCLERK	800.00
2	ALLENSALESMAN	1600.00
3	WARDSALESMAN	1250.00

Nun versuchen wir ein Wert in dert Spalte ENAME zu ändern:

```
UPDATE emp_function_demo_view SET ENAME = 'Test1' WHERE ENAME = 'SMITHCLERK';
```

Fehler: `virtual column not allowed here`

3.2.2 Join Views

Als erstes erstellen wir eine entsprechende View:

```
CREATE OR REPLACE VIEW emp_dept_view
            (empno, ename, dname)
AS SELECT empno, ename, dname
   FROM EMP JOIN DEPT USING (deptno);
```

Grundsätzlich gilt, wir können nur Daten der Tabelle änderen welche `key-preserved` ist, dass heisst in dieser Tabelle wo der Primarykey erhalten bleibt. In unserem Fall ist das EMP. Aus diesem Grund funktioniert der folgende Query auch problemlos:

```
UPDATE emp_dept_view SET ENAME = 'TESTER' WHERE DNAME = 'RESEARCH';
```

	EMPNO	ENAME	DNAME
1	7369	TESTER	RESEARCH
2	7499	ALLEN	SALES
3	7521	WARD	SALES
4	7566	TESTER	RESEARCH

Wollen wir hingeben auf `DEPT` etwas ändern:

```
UPDATE emp_dept_view SET DNAME = 'TESTER' WHERE EMPNO = '7369';
```

Fehler: `cannot modify a column which maps to a non key-preserved table`
(Diese Einschränkung gilt auch wenn eine 1-1 Beziehung Vorhanden ist)

3.3 WITH CHECK OPTION

Ohne die `CHECK OPTION` ist es möglich, dass ein User Daten aus der View "verschwinden" lassen kann - er kann Änderungen durchführen welche in der View nicht sichtbar sind. Um dies zu demonstrieren erstellen wir zuerst zwei Views ohne `CHECK OPTION`:

```
-- create views
CREATE OR REPLACE VIEW sal_gt_1000_view
AS SELECT * FROM EMP WHERE SAL > 1000;

CREATE OR REPLACE VIEW sal_gt_1000_and_dept_eq_30_view
AS SELECT * FROM sal_gt_1000_view WHERE DEPTNO = 30;

-- Show result
SELECT * FROM sal_gt_1000_and_dept_eq_30_view;
```

	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	7521	WARD	SALESMAN	7698	1981-02-22 00:00:00	1250.00	500.00	30
2	7654	MARTIN	SALESMAN	7698	1981-09-28 00:00:00	1250.00	1400.00	30
3	7698	BLAKE	MANAGER	7839	1981-05-01 00:00:00	2850.00	<null>	30
4	7844	TURNER	SALESMAN	7698	1981-09-08 00:00:00	1500.00	0.00	30

Wir lassen nun demonstrativ einen Mitarbeiter "verschwinden" indem bei *WARD* das Department geändert wird:

```
UPDATE sal_gt_1000_and_dept_eq_30_view SET DEPTNO = 20 WHERE EMPNO = 7521;
SELECT * FROM sal_gt_1000_and_dept_eq_30_view;
```

	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	7654	MARTIN	SALESMAN	7698	1981-09-28 00:00:00	1250.00	1400.00	30
2	7698	BLAKE	MANAGER	7839	1981-05-01 00:00:00	2850.00	<null>	30
3	7844	TURNER	SALESMAN	7698	1981-09-08 00:00:00	1500.00	0.00	30

Setzen wir nun auf der `sal_gt_1000_and_dept_eq_30_view` die `CHECK OPTION`

```
CREATE OR REPLACE VIEW sal_gt_1000_and_dept_eq_30_view
AS SELECT * FROM sal_gt_1000_view WHERE DEPTNO = 30 WITH CHECK OPTION;
-- Try again
UPDATE sal_gt_1000_and_dept_eq_30_view SET DEPTNO = 20 WHERE EMPNO = 7654;
```

Fehler: `view WITH CHECK OPTION where-clause violation`

Untersuchen wir auch noch den Fall wenn die `CHECK OPTION` nur auf `sal_gt_1000_view` gesetzt ist:

```
-- remove check option from combined view
CREATE OR REPLACE VIEW sal_gt_1000_and_dept_eq_30_view
AS SELECT * FROM sal_gt_1000_view WHERE DEPTNO = 30;
```

```
CREATE OR REPLACE VIEW sal_gt_1000_view
AS SELECT * FROM EMP WHERE SAL > 1000 WITH CHECK OPTION;

-- Try again
UPDATE sal_gt_1000_and_dept_eq_30_view SET DEPTNO = 20 WHERE EMPNO = 7654;
```

	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	7698	BLAKE	MANAGER	7839	1981-05-01 00:00:00	2850.00	<null>	30
2	7844	TURNER	SALESMAN	7698	1981-09-08 00:00:00	1500.00	0.00	30

Der vorherige Update funktioniert nun ohne Probleme. Die Schlussfolgerung daraus ist, `CHECK OPTION` funktioniert nur innerhalb ihrer view!

4 Zugriffssteuerung mit User und Rollen

4.1 View erstellen

```
CREATE OR REPLACE VIEW emp_summary_view
AS SELECT ename, job, dname, loc
FROM  EMP JOIN DEPT D on EMP.DEPTNO = D.DEPTNO;

-- show result
SELECT * FROM emp_summary_view;
```

	ENAME	JOB	DNAME	LOC
1	CLARK	MANAGER	ACCOUNTING	NEW YORK
2	KING	PRESIDENT	ACCOUNTING	NEW YORK
3	MILLER	CLERK	ACCOUNTING	NEW YORK
4	TESTER	CLERK	RESEARCH	DALLAS
5	ALLEN	SALESMAN	RESEARCH	DALLAS
6	TESTER	MANAGER	RESEARCH	DALLAS
7	TESTER	ANALYST	RESEARCH	DALLAS
8	TESTER	CLERK	RESEARCH	DALLAS
9	TESTER	ANALYST	RESEARCH	DALLAS

4.2 Rollen definieren

```
CREATE ROLE emp_user_role;
CREATE ROLE emp_manager_role;
```

4.3 Den Rollen Rechte zuweisen

```
GRANT SELECT ON emp_summary_view TO emp_user_role;
GRANT SELECT,UPDATE,INSERT,ALTER,DELETE ON EMP TO emp_manager_role;
GRANT CREATE VIEW TO emp_manager_role;
```

4.4 Den User Rollen zuweisen

```
GRANT emp_user_role TO charlie;
GRANT emp_manager_role TO snoopy;
```

4.5 Überprüfen Sie die beiden Rollen

4.5.1 User charlie:

```
SELECT * FROM SCOTT.emp_summary_view;
```

	ENAME	JOB	DNAME	LOC
1	CLARK	MANAGER	ACCOUNTING	NEW YORK
2	KING	PRESIDENT	ACCOUNTING	NEW YORK
3	MILLER	CLERK	ACCOUNTING	NEW YORK
4	TESTER	CLERK	RESEARCH	DALLAS
5	ALLEN	SALESMAN	RESEARCH	DALLAS
6	TESTER	MANAGER	RESEARCH	DALLAS

```
UPDATE SCOTT.EMP_SUMMARY_VIEW SET ENAME = 'Evil_c' WHERE ENAME = 'TESTER';
```

Fehler: `insufficient privileges`

4.5.2 User snoopy:

Versuchen wir das gleiche Update nun mit dem user snoopy:

```
UPDATE SCOTT.EMP SET ENAME = 'Evil_c' WHERE ENAME = 'TESTER';
SELECT * FROM SCOTT.EMP;
```

Output | SCOTT.EMP

14 rows

	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO	FIRE
1	7369	Evil_c	CLERK	7902	2020-03-09 09:24:34	800.00	<null>	20	<null>
2	7499	ALLEN	SALESMAN	7698	1981-02-20 00:00:00	1600.00	300.00	20	<null>
3	7521	WARD	SALESMAN	7698	1981-02-22 00:00:00	1250.00	500.00	30	<null>
4	7566	Evil_c	MANAGER	7839	1981-04-02 00:00:00	2975.00	<null>	20	<null>
5	7654	MARTIN	SALESMAN	7698	1981-09-28 00:00:00	1250.00	1400.00	30	<null>

5 Zugriffsrechte: Objekt- und Systemrechte

5.1 Objektrechte

5.1.1 Was sind Objektrechte?

Sie berechtigen dazu, Operationen auf existierenden Objekten auszuführen. Die Rechte können einem Datenbankbenutzer oder einer Rolle zugeteilt werden können, um so gewisse Operationen auf der Objektebene auszuführen. Diese Operationen können `SELECT` , `INSERT` , `UPDATE` , `DELETE` , `ALTER` , `INDEX` auf Tabellen und Views oder `EXECUTE` auf Prozeduren und Fuktionen sein. Ebenso gibt es die Möglichkeit, einzelne Operationen auszunehmen.

5.1.2 Das Experiment

In diesem Experiment wollen wir untersuchen, ob weitergegebene Objektrechte kaskadierend entfernt werden. Dazu legen wir uns als `system` zwei neue Benutzer `mickey` und `minnie` an:

```
CREATE USER mickey IDENTIFIED BY mickeyInDisneyLand;
-- User MICKEY erstellt.

GRANT CREATE SESSION TO mickey;
-- Grant erfolgreich.


CREATE USER minnie IDENTIFIED BY minnieInDisneyLand;
-- User MINNIE erstellt.
```

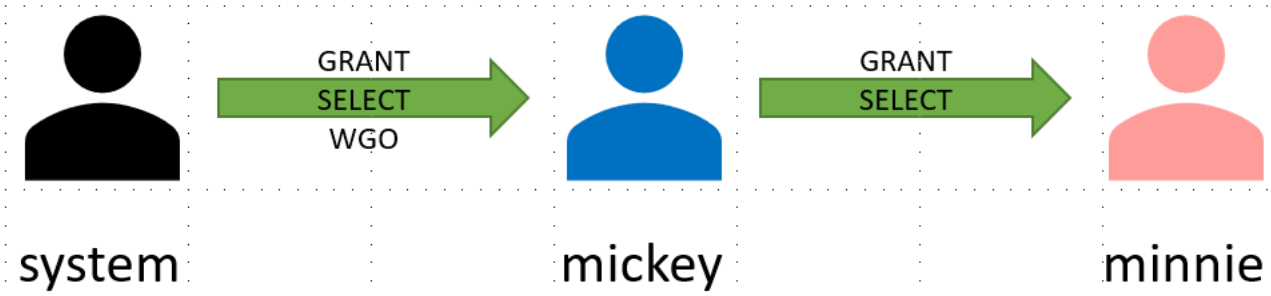
```
GRANT CREATE SESSION TO minnie;
-- Grant erfolgreich.
```

Nun geben wir `mickey` die Berechtigung, `SELECT`-Statements auf der Tabelle `emp` auszuführen und diese auch weiterzugeben:

```
GRANT SELECT ON scott.emp TO mickey WITH GRANT OPTION;
-- Grant erfolgreich.
```

Als `mickey` geben wir die `SELECT`-Berechtigung an `minnie` weiter:

```
GRANT SELECT ON scott.emp TO minnie;
-- Grant erfolgreich.
```



Mit dem folgenden SQL-Statement lassen sich die Berechtigungseinträge für die Tabelle `emp` anzeigen:

```
SELECT * FROM USER_TAB_PRIVS WHERE TABLE_NAME = 'EMP';
```

	GRANTEE	OWNER	TABLE_NAME	GRANTOR	PRIVILEGE	GRANTABLE	HIERARCHY	COMMON	TYPE	INHERITED
1	MICKEY	SYSTEM	EMP	SYSTEM	SELECT	YES	NO	NO	TABLE	NO
2	MINNIE	SYSTEM	EMP	MICKEY	SELECT	NO	NO	NO	TABLE	NO

Der Benutzer `system` entzieht nun `mickey` die zuvor erteilte Berechtigung mit:

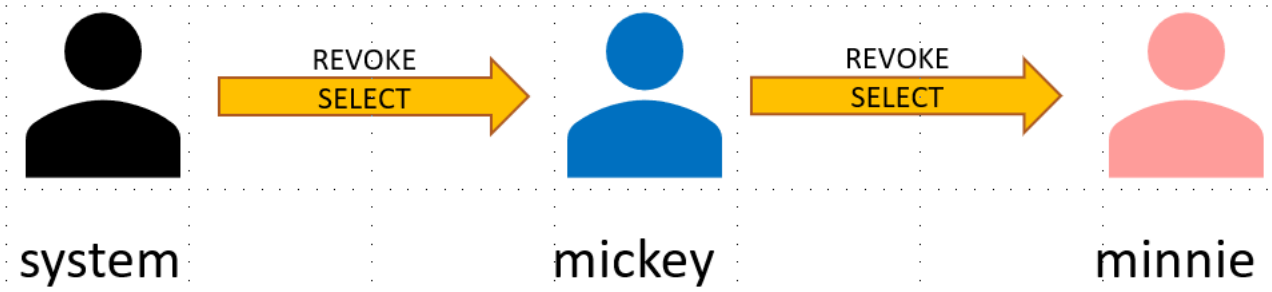
```
REVOKE SELECT ON scott.emp FROM mickey;
-- Revoke erfolgreich.
```

5.1.3 Beobachtung

Schauen wir uns jetzt nochmals die Berechtigungseinträge mit dem obigen Statement an, so stellen wir fest das keine Einträge mehr vorhanden sind.

GRANTEE	OWNER	TABLE_N...	GRANTOR	PRIVILEGE	GRANTABLE	HIERARCHY	COMMON	TYPE	INHERITED

Die mit `GRANT OPTION` verteilten Berechtigungen werden also kaskadierend widerrufen.



5.2 Systemrechte

5.2.1 Was sind Systemrechte?

Sie berechtigen zu Schemaoperationen konkret geht es hier um die Operationen `CREATE` , `ALTER` , `DROP` und die bereits bekannten Objektoperationen. Diese Rechte können analog zu den Objektrechten pro Benutzer oder systemweit verteilt oder widerrufen werden. Zugriffe und Veränderungen an den eigenen Tabellen werden nicht explizit aufgeführt, es ist bereits durch die Architektur gegeben.

5.2.2 Das Experiment

In diesem Experiment wollen wir untersuchen, ob weitergegebene Systemrechte kaskadierend entfernt werden. Dazu greifen wir auf die zuvor von `system` angelegten Benutzer `mickey` und `minnie` zurück.

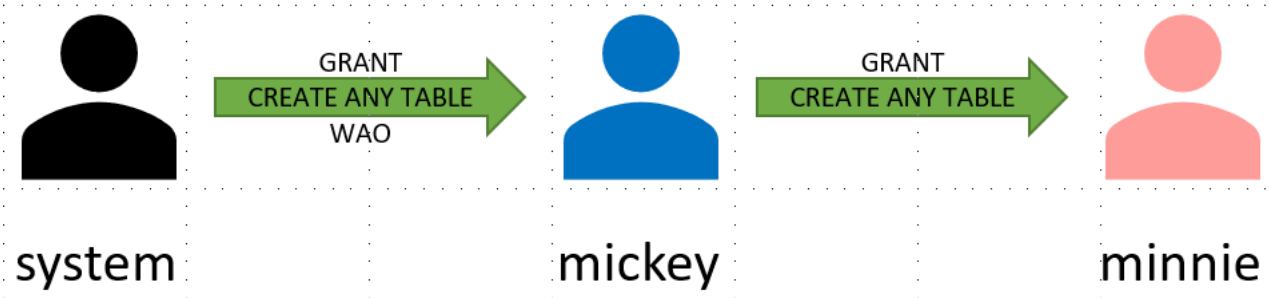
Nun geben wir `mickey` die Berechtigung, in jedem Schema Tabellen zu erzeugen und diese auch weiterzugeben:

```
GRANT CREATE ANY TABLE TO mickey WITH ADMIN OPTION;
-- Grant erfolgreich.
```

Als `mickey` erstellen wir eine Tabelle und geben diese `CREATE`-Berechtigung an `minnie` weiter:

```
CREATE TABLE mickey_friends(
    friend_id NUMBER,
    first_name VARCHAR2(50) NOT NULL,
    last_name VARCHAR2(50) NOT NULL,
    PRIMARY KEY(friend_id)
);
-- Table MICKEY_FRIENDS erstellt.

GRANT CREATE ANY TABLE TO minnie;
-- Grant erfolgreich.
```



Als `minnie` erstellen wir ebenfalls eine Tabelle:

```
CREATE TABLE minnie_friends(
    friend_id NUMBER,
    first_name VARCHAR2(50) NOT NULL,
    last_name VARCHAR2(50) NOT NULL,
    PRIMARY KEY(friend_id)
);
-- Table MINNIE_FRIENDS erstellt.
```

Mit dem folgenden SQL-Statement lassen sich die Systemrechte für `mickey` und `minnie` anzeigen:

```
SELECT * FROM SYS.DBA_SYS_PRIVS WHERE GRANTEE = 'MICKEY' OR GRANTEE = 'MINNIE';
```

	GRANTEE	PRIVILEGE	ADMIN_OPTION	COMMON	INHERITED
1	MICKEY	CREATE ANY TABLE	YES	NO	NO
2	MINNIE	CREATE TABLE	NO	NO	NO
3	MINNIE	CREATE SESSION	NO	NO	NO
4	MICKEY	CREATE TABLE	YES	NO	NO
5	MICKEY	CREATE SESSION	NO	NO	NO
6	MINNIE	CREATE ANY TABLE	NO	NO	NO

Der Benutzer `system` entzieht nun `mickey` die zuvor erteilte Berechtigung mit:

```
REVOKE CREATE ANY TABLE FROM mickey;
-- Revoke erfolgreich.
```

5.2.3 Beobachtung

Schauen wir uns jetzt nochmals die Systemrechte mit dem obigen Statement an:

	GRANTEE	PRIVILEGE	ADMIN_OPTION	COMMON	INHERITED
1	MINNIE	CREATE TABLE	NO	NO	NO
2	MINNIE	CREATE SESSION	NO	NO	NO
3	MICKEY	CREATE SESSION	NO	NO	NO
4	MINNIE	CREATE ANY TABLE	NO	NO	NO

Anderst als bei den Objektrechten ist hier kein kaskadierendes Verhalten erkennbar, wenn man Systemrechte für `mickey` widerruft, bleiben jene von `minnie` bestehen. Es hat keinen Einfluss, ob die Berechtigung mittels `ADMIN OPTION` erteilt wurde.

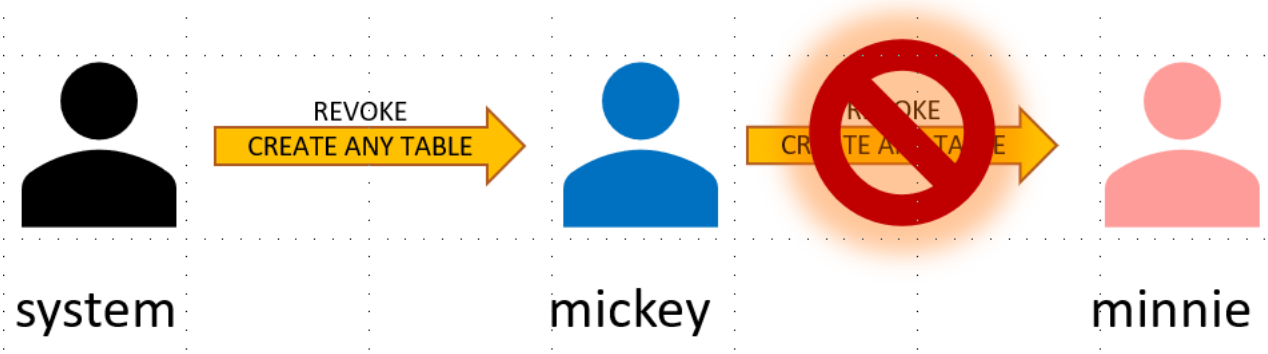
Mit dem folgenden SQL-Statement überprüfen wir zusätzlich, ob die jeweils erstellten Tabellen noch existieren:

```
SELECT OWNER, TABLE_NAME FROM ALL_TABLES WHERE OWNER = 'MICKEY' OR OWNER = 'MINNIE';
```

	OWNER	TABLE_NAME
1	MINNIE	MINNIE_FRIENDS
2	MICKEY	MICKEY_FRIENDS

Die Tabellen existieren nach wie vor und wurden nicht mit den Systemrechten zusammen entfernt.

Die mit `ADMIN OPTION` verteilten Berechtigungen werden also **nicht** kaskadierend widerrufen:



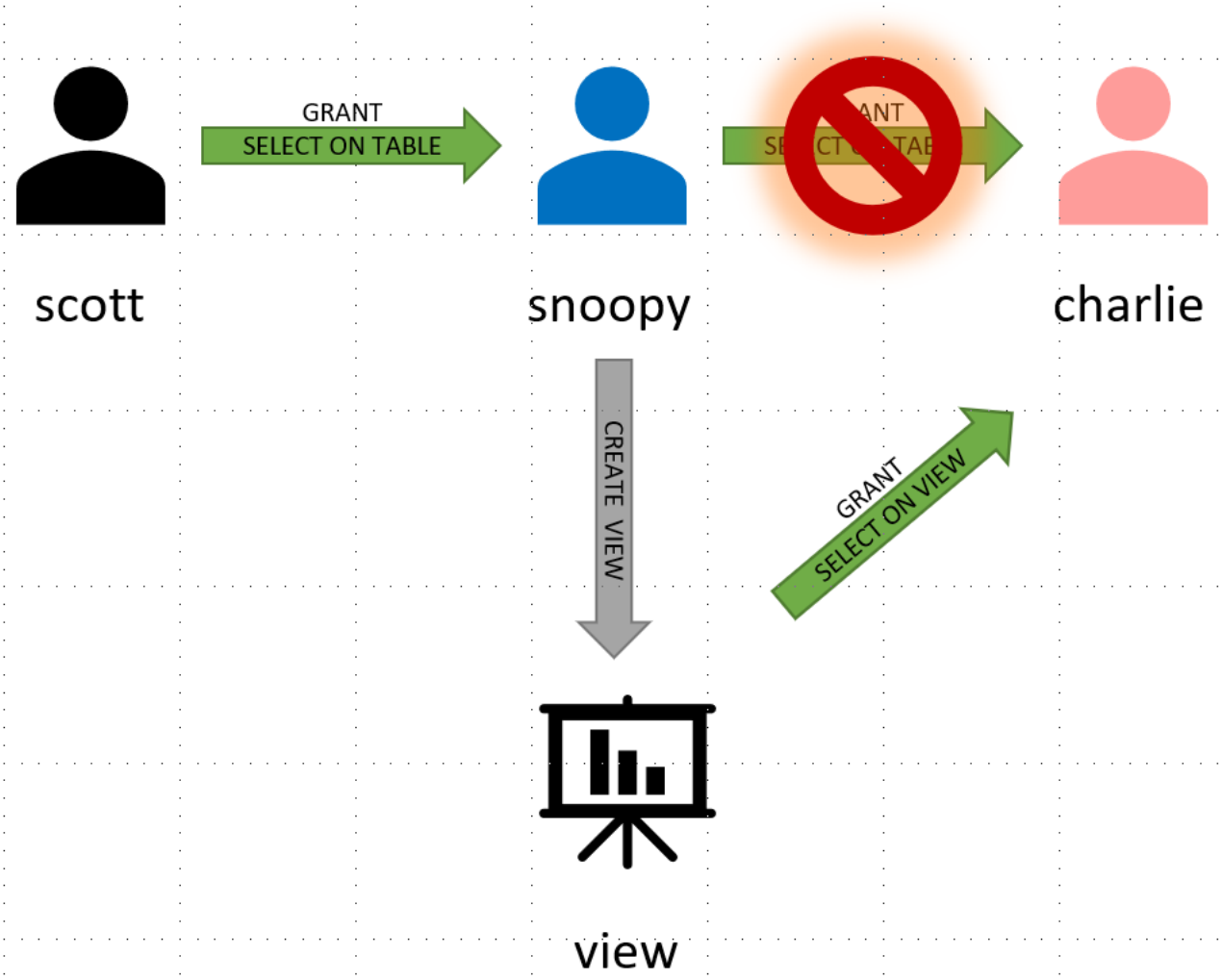
5.3 Rechte auf Views

5.3.1 Szenario

Der Benutzer `scott` ermöglicht dem Benutzer `snoopy` Lesezugriff auf die `EMP`-Tabelle allerdings ohne die `WITH GRANT OPTION`, damit kann `snoopy` die Berechtigung nicht weitergeben.

5.3.2 Das Experiment

Der Benutzer `charlie` möchte nun über `snoopy` auf die `EMP`-Tabelle zugreifen. Da `snoopy` die Berechtigung hat, neue Views zu erstellen, verfolgen wir diesen Ansatz weiter.



Wir erstellen mit dem Benutzer `snoopy` eine neue View für die `EMP`-Tabelle. Damit das Funktioniert müssen wir allerdings noch etwas Vorarbeit leisten Zitat Oracle-Doc:

> `The owner of the schema containing the view must have the privileges necessary to either select, insert, update, or delete rows from all the tables or views on which the view is based. The owner must be granted these privileges directly, rather than through a role`
>
> –*https://docs.oracle.com/cd/B28359_01/server.111/b28286/statements_8004.htm*

```
GRANT SELECT on scott.emp TO snoopy;

CREATE OR REPLACE VIEW EMP_FOR_CHARLIE_VIEW AS SELECT EMPNO, ENAME, JOB, HIREDATE FROM SCOTT.EMP;
```

Anschliessend geben wir diese View mit den selben Berechtigungen an `charlie` weiter:

```
GRANT SELECT,INSERT,DELETE,UPDATE ON snoopy.EMP_FOR_CHARLIE_VIEW TO charlie;
-- GRANT erfolgreich.
```

5.3.3 Beobachtung

Damit lässt dich die `GRANT OPTION` umgehen und die Berechtigungen weitergeben, allerdings müssen Referenzen auf Views separat erlaubt werden.