

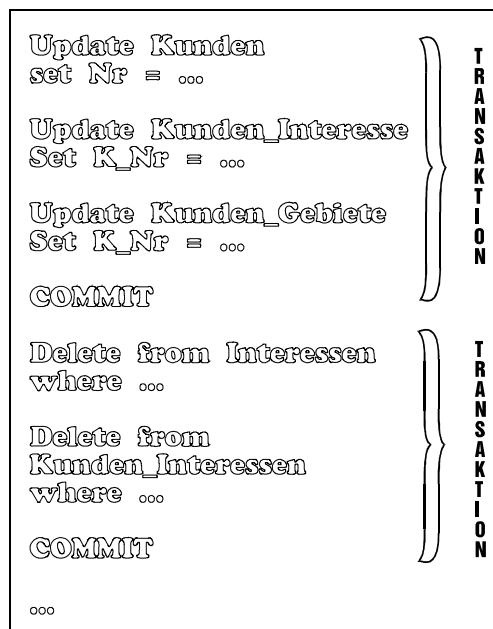
7 DCL (Data Control Language)

In diesem Kapitel werden wir die Syntax der SQL-Befehle für Transaktionen und für die Vergabe von Privilegien kennen lernen.

Eine der wichtigsten Eigenschaften einer Datenbank besteht in der Mehrplatz-Fähigkeit. Wie man sich leicht vorstellen kann, ist der gemeinsame Zugriff von mehreren Anwendern auf die gleichen Daten problematisch und erfordert deshalb eine besondere Behandlung. Die Mehrplatz-Fähigkeit führt jedoch zu einem neuen Problem, nämlich zum Datenschutz und der daraus folgenden Privilegienvergabe.

7.1 Behandlung von Transaktionen

In Datenbanksprachen werden Befehlsfolgen in logische Bereiche (Transaktionen) zusammengefasst. Eine Transaktion beginnt mit dem ersten SQL-Befehl oder der Anweisung BEGIN TRANSACTION und endet mit einem COMMIT (TRANSACTION) oder einem ROLLBACK, dann beginnt die nächste Transaktion usw. Alle Befehle einer Transaktion sollten zusammengehören. Im Falle eines Systemausfalls (Platte voll, Stromausfall, Hardware-Fehler...) garantiert die Datenbank, dass entweder die Transaktion als ganzes durchgeführt werden kann oder überhaupt nicht. Dies garantiert eine konsistente Datenbank!



7.1.1 COMMIT - Statement (Transaktion abschließen)

Mit dem SQL-Befehl COMMIT kann man eine Transaktion erfolgreich abschließen. In jedem "SQL-Programm", in dem Daten geändert werden, ist die Verwendung von COMMIT notwendig.

Jede Veränderung von Daten ist dem ausführenden Benutzer sofort nach der Ausführung sichtbar: Allen anderen Benutzern ist jedoch eine Transaktion nur nach der Ausführung vom COMMIT sichtbar.

Syntax

```
COMMIT [WORK | TRANsaction]
```

Der Zusatz „Work“ bzw. „Transaction“ ist im ANSI SQL notwendig. Die meisten Datenbanksysteme erlauben jedoch, einfach ein COMMIT anzugeben. „COMMIT“ und „COMMIT WORK“ bewirken dasselbe.

Beispiel

Wenn wir uns z.B. von dem Kunden 'MARKUS' trennen müssen, so lässt sich dies in der Datenbankterminologie so formulieren:

```
BEGIN TRANSACTION
DELETE FROM Kunden WHERE Nr = 3
DELETE FROM Kunden Gebiete WHERE K Nr = 3
DELETE FROM Kunden Interessen WHERE K Nr = 3
COMMIT TRANSACTION
...
```

Kommt nach dem ersten DELETE ein Systemausfall, so führt die Datenbank überhaupt keine Änderung der Daten durch.

7.1.2 ROLLBACK - Statement (Transaktion verwerfen)

Im Gegensatz zum COMMIT - Befehl, wird der ROLLBACK - Befehl viel seltener verwendet. Mit einem ROLLBACK - Befehl kann man alle Transaktionen vom letzten COMMIT oder ROLLBACK verwerfen.

Syntax

```
ROLLBACK [WORK | TRANsaction]
[SAVEPOINT <savepointname>]
```

Die Verwendung von einem Savepoint ermöglicht, eine feinere Transaktion darzustellen. Dann erfolgt das Zurücksetzen des Datenbestandes nur bis zum angegebenen Punkt im Programm.

Wir möchten an dieser Stelle den Leser darauf aufmerksam machen, dass die Verwendung von Savepoints nicht anzuraten ist, weil sie der Übersicht schadet.

7.2 Mehrfachzugriff auf Tabellen und Sperrstrategien

Achtung: Das explizite Sperren von Tabellen ist nicht ANSI - Standard. D.h. die hier vorgestellten Befehle sind nur als eine Möglichkeit und nicht als ANSI - Syntax zu betrachten. Unabhängig von einer expliziten Sperre werden die Datensätze bei jedem UPDATE von allen relationalen Datenbanksystemen bis zu einem Commit automatisch gesperrt.

Beispiel :

Wir erweitern unser Übungsmodell um eine Tabelle: REISEN. In dieser Tabelle stehen alle Gruppenreisen und die vorhandenen Plätze, die das Reisebüro anbietet.

REISEN	NR	BEZ	VON	BIS	PLÄTZE
	1	Rom	23.03.1993	03.04.1993	3

Unser Reisebüro beschäftigt zwei Angestellte, die den Kunden beraten und dabei ein Informationsprogramm verwenden. Beide Berater haben gleichzeitig zwei Kunden, die sich für eine Reise nach Rom interessieren. Beide Kunden entscheiden sich für die Reise und möchten zwei bzw. drei Plätze reservieren. Insgesamt sind also fünf Plätze notwendig, vorhanden sind jedoch nur drei. Die zwei Angestellten des Reisebüros wissen jedoch von dem Zufall nichts und führen die Buchung aus!

Berater 1	Berater 2
Bestimmt freie Plätze -> 3	Bestimmt freie Plätze -> 3
Bucht : freie Plätze = 3 - 2 Speichert die Zahl 1 unter freie Plätze	
	Bucht : freie Plätze = 3 - 3 Speichert die Zahl 0 unter freie Plätze

Wir sehen, das Reisebüro würde wahrscheinlich viele Kunden verlieren, falls dieses EDV-System verwendet würde. Das Problem lässt sich jedoch lösen, indem die Tabelle oder nur die betreffenden Datensätze vor der Änderung gesperrt und nach der Änderung wieder freigegeben werden.

Eine gesperrte Tabelle wird mit COMMIT oder ROLLBACK wieder freigegeben!

Im folgenden lernen wir drei verschiedenen Arten des Sperrens. Wir möchten hier keinen Anspruch auf Vollständigkeit erheben.

Das Sperren von Datensätze ist nicht im ASCII-SQL enthalten. Die hier angebotene Lösung ist nur eine Möglichkeit von vielen.

7.2.1 SHARE LOCK

Mit SHARE LOCK kann die Tabelle für das Verändern und Löschen von Datensätzen gesperrt werden. Die Tabelle kann von mehreren Benutzern gleichzeitig im SHARE Modus gesperrt werden. Sie kann wieder verändert und gelöscht werden, wenn alle Benutzer, die die Tabelle im SHARE Modus gesperrt haben, sie freigeben.

Die übliche Syntax für ein SHARE LOCK:

```
LOCK TABLE [<User>.<tablename>] IN SHARE [nowait]
```

Diese Syntax wird nicht von allen Datenbanksystemen unterstützt!

Achtung: SHARE LOCK ist nur selten wirklich notwendig und ist hier nur der Vollständigkeit halber erwähnt. Das obige Beispiel kann man mit dieser Methode nicht lösen.

7.2.2 EXCLUSIVE LOCK

Mit EXCLUSIVE LOCK kann man ganze Tabellen für alle anderen Benutzer sperren. Nach dem Sperren können die gesperrten Tabellen nur vom Sperrenden verändert und gelöscht, von allen anderen aber nur gelesen werden. Diese Art des Sperrens ist zwar sehr sicher und auch einfach in der Anwendung, führt aber in vielen Applikationen zu großen Wartezeiten für alle anderen Benutzer und wird deshalb nicht allzu oft verwendet.

Die übliche Syntax für ein EXCLUSIVE LOCK:

```
LOCK TABLE [<User>.<tablename>] IN EXCLUSIVE [nowait]
```

Diese Syntax wird nicht von allen Datenbanksystemen unterstützt!

7.2.3 EXCLUSIVE LOCK für Zeilen

Das Sperren von einzelnen Zeilen ist sicherlich die beste Möglichkeit, das obige Problem zu lösen. Im Allgemeinen empfehlen wir die Verwendung der hier vorgestellten Methode, falls das Sperren von Daten notwendig ist. Manche DB-Systeme (z.B. MS SQL-Server) erlauben jedoch keine Zeilenspernung.

Wie die Überschrift schon besagt, kann man auch einzelne Zeilen einer Tabelle ausschließlich sperren. Das bedeutet, dass die gesperrten Zeilen von keinem anderen Benutzer verändert oder gelöscht werden können, bis der Sperrende die Zeilen mit COMMIT oder ROLLBACK freigibt.

Syntax

```
SELECT ....  
FROM ...  
[WHERE ...]  
[ORDER BY ...]  
FOR UPDATE OF <Spalte1> [, <Spalte2>] ...
```

Achtung: Wenn man die FOR UPDATE OF-Klausel verwendet, darf die GROUP BY -Klausel nicht angewandt werden.

Beispiel

```
SELECT Bez, Von, Bis, Plaetze  
FROM Reisen  
WHERE Bez = 'Rom'  
FOR UPDATE OF Plaetze  
UPDATE Reisen  
SET Plaetze = Plaetze -  
      Anzahl_der_zu_reservierenden_Plaetze  
WHERE Bez = 'Rom'  
Commit
```

Wir lösen das obige Beispiel:

Anzahl_der_zu_reservierenden_Plaetze wäre im ersten Fall 2, im zweiten Fall 3.

7.3 Weitere Lock-Methoden

Einige DB-Systeme bieten die Möglichkeit mittels dem SET - Befehl, die „Lesbarkeit“ von Tabellen zu beeinflussen:

Syntax

```
SET transaction isolation level
    READ COMMITTED |
    READ UNCOMMITTED |
    REPEATABLE READ |
    SERIALIZABLE
```

- **READ COMMITTED:**
Es können nur jene Datensätze gelesen werden, die schon mit COMMIT abgeschlossen wurden. In dieser Stufe sind keine nichtwiederholbaren Lesevorgänge (so genannte „Dirty Reads“) möglich. Phantomwerte sind sehr wohl möglich!
- **READ UNCOMMITTED:**
Es können auch jene Datensätze gelesen werden, die **noch nicht** mit COMMIT abgeschlossen wurden. „Dirty Reads“ **und** Phantomwerte sind möglich.
- **REPEATABLE READ oder SERIALIZABLE:**
Es sind weder „Dirty Reads“ noch Phantomwerte möglich. Diese Einstellung sollte jedoch nur in ganz besonderen Fällen verwendet werden, da die betroffenen Tabellen während der Transaktion gesperrt sind.

Manche DB-Systeme erlauben das Überschreiben der Sperranweisung auch innerhalb des SELECT - Statements.

7.4 Privilegienbehandlung

Von Datenschutzverletzungen kann schon mancher ein Lied singen. In der EDV gehört dieses Problem sicherlich zu den heikelsten Themen überhaupt. Durch eine richtige Privilegienvergabe, die nur bestimmten Benutzern bestimmte Daten zur Verfügung stellt, kann sich ein Betrieb in einigen Fällen eine Gerichtsklage oder andere Unannehmlichkeiten ersparen.

Man kann Privilegien aus zwei verschiedenen Sichten vergeben.

- Aus der Sicht des Benutzers
- Aus der Sicht von Tabellen oder Views

Beide Formen der Privilegienvergabe erfolgen mit dem GRANT - Statement.

7.4.1 Das GRANT - Statement (für Benutzer)

Eine effiziente Art, Daten und Datenstrukturen vor unerlaubtem Zugriff zu schützen, ist das Unterteilen von Benutzern in Anwendergruppen.

Syntax

```
GRANT { RESOURCE | DBA }  
      TO <User1> [, <User2> ...]  
  
oder  
  
GRANT CONNECT  
      TO <User1> [, <User2> ...]  
      identified by <Password1> [, <Password2> ...]
```

Es besteht die Möglichkeit, gleich mehrere Anwender mit dem GRANT Befehl zu bearbeiten.

Bei Anlegen eines Anwenders (CONNECT) sollte auch ein Passwort mitgegeben werden.

In Relationalen Datenbanken werden oft folgende Arten von Anwendergruppen unterschieden (Anwendergruppen sind nicht ANSI normiert!).

Gruppe	Beschreibung
CONNECT	<p>Erlaubt sind alle DML - Operationen mit Ausnahme jener Einschränkungen, die auf der Tabellen- oder Viewebene gelten.</p> <p>Die CONNECT - Rechte erlauben dem Benutzer den Zutritt zur Datenbank und müssen vor allen anderen Rechten vergeben werden. Sie sind die einfachsten Rechte, die in einer Datenbank möglich sind und sind für Applikationsanwender gedacht.</p>
RESOURCE	<p>Erlaubt sind die Rechte von CONNECT, alle Operationen der DDL für eigene Tabellen und die Rechtevergabe (REVOKE) für die eigenen Tabellen.</p> <p>Der Benutzer besitzt <i>alle</i> Rechte einer selbst angelegten Tabelle!</p> <p>Die RESOURCE - Rechte sind für den Programmierer gedacht, sie garantieren ein kollisionsloses Arbeiten in einer Mehrplatz-Datenbank.</p>
DBA	<p>Erlaubt sind die Rechte von RESOURCE, jedoch nicht nur für die eigenen Tabellen, sondern auch für alle anderen. Das Einfügen oder Ändern von Benutzerprivilegien ist nicht erlaubt.</p> <p>Die DBA-Rechte sind, wie der Name schon verrät, für den Datenbankadministrator gedacht.</p>
SYSADM oder SA	<p>Erlaubt sind <i>alle</i> in einer Datenbank möglichen Rechte, d.h. auch das Anlegen/Löschen/Verändern von Anwender.</p> <p>Ein SYSADM ist schon von Anfang an in jeder Datenbank vorhanden, diese Rechte können mit dem GRANT - Befehl weder angelegt noch verändert werden. Man kann (was auch empfehlenswert ist) mit dem GRANT - Befehl nur das Passwort ändern.</p>

Beispiel

Wir möchten drei Benutzer anlegen. Alex soll die dba - Rechte, Johannes die RESOURCE - Rechte und Andreas die CONNECT - Rechte erhalten.

```
GRANT CONNECT
    TO alex, johannes, andreas
    IDENTIFIED BY alex, johannes, andreas

GRANT RESOURCE
    TO johannes

GRANT DBA
    TO alex
```

7.4.2 Das GRANT - Statement (für Tabellen)

Die bis jetzt vorgestellte Rechtevergabe reicht sicherlich nicht, um einen intakten Datenschutz zu garantieren. Deshalb besteht die Möglichkeit, jede Tabelle oder jeden VIEW auf nur ausgewählte Benutzer zu beschränken.

Syntax

```
GRANT { <Privileg1> [, <Privileg1> ..] } | ALL
ON [<User1>.<Objekt>]
TO {<User2> [, <User3> ...]} | PUBLIC
```

Folgende Privilegien sind möglich:

ALL

Alle möglichen Rechte auf die Tabelle oder den VIEW sind erlaubt.

SELECT

Lesen in der Tabelle oder im VIEW ist erlaubt.

INSERT

Einfügen in eine Tabelle oder einen VIEW ist erlaubt.

DELETE

Löschen in einer Tabelle oder einem VIEW ist erlaubt.

INDEX

Erzeugen von Indizes für eine Tabelle ist erlaubt.

ALTER

Tabellenstruktur verändern ist erlaubt.

UPDATE oder UPDATE (Spalte ,[Spalte...])

Verändern von Daten in einer Tabelle ist erlaubt.

Wenn man bei der Rechtevergabe einzelne Spalten angibt, so können nur die Daten der angegebenen Spalten verändert werden.

Ein Objekt ist eine Tabelle oder ein VIEW.

Mit der Option PUBLIC können die Rechte einer Tabelle oder eines Views für alle Datenbankbenutzer vergeben werden.

Beispiel

Die Tabelle Kunden soll für alle Benutzer zugänglich sein.

```
GRANT ALL
      ON Kunden
      TO PUBLIC
```

Beispiel

Die Tabelle Interessen soll für Andreas nur lesbar sein.

```
GRANT SELECT
      ON Interessen
      TO andreas
```

Beispiel

Die Tabelle Reisen soll für Andreas lesbar und für die Spalte "Plätze" veränderbar sein.

```
GRANT SELECT, UPDATE(Plätze)
      ON Reisen
      TO andreas
```

7.4.3 Das REVOKE - Statement (Aufgabe von Privilegien)

In einem Software-Projekt ist die Fluktuation von Mitarbeitern sehr hoch. Vergebene Rechte sollten deshalb bei Abgang eines Mitarbeiters im Sinne des Datenschutzes wieder entzogen werden. Dies lässt sich mit dem Befehl REVOKE erreichen.

Auf Benutzerebene:

Syntax

```
revoke {CONNECT | RESOURCE | DBA }  
FROM <User1> [, <User2> ...]
```

Auf Tabellen- oder Viewebene:

Syntax

```
revoke {ALL | SELECT | INSERT | DELETE | INDEX | ALTER |  
        UPDATE | UPDATE (<Spalte1> [, <Spalte2> ...]}  
ON {<Tabellenname> | <Viewname>}  
FROM {PUBLIC | <User1> [, <User2> ...]}
```

Beispiel

Andreas darf die Tabelle Interessen nicht mehr lesen.

```
revoke SELECT  
        ON Interessen  
        FROM andreas
```

7.5 Verbindung zur Datenbank

Die Verbindung zur Datenbank ist der erste Schritt, wenn man eine Datenbank in Anspruch nehmen möchte. Bei der Verwendung einer SHELL geschieht dies normalerweise beim Aufruf der SHELL. Zu diesem Zeitpunkt muss der potentielle Datenbanknutzer zumindest die CONNECT - Privilegien besitzen.

Wenn man aus einer anderen Sprache SQL verwenden oder in einer SQL-SHELL einen User wechseln möchte, kann das CONNECT- und DISCONNECT - Statement angewendet werden.

7.5.1 Das CONNECT - Statement

Verbindung zu einer Datenbank herstellen.

Syntax

```
CONNECT [<Datenbankname>] <cursor-number> [<User>  
[/  
<Password>]]
```

Cursor-number ist eine eindeutige Zahl, die eine Verbindung zu einer bestimmten Datenbank ermöglicht. Besonders dann, wenn eine Programmiersprache verwendet wird, können mehrere Datenbanken gleichzeitig verwendet werden.

Diese Option ist jedoch nicht in allen Datenbanken möglich!

7.5.2 Das DISCONNECT - Statement

Verbindung zu einer Datenbank abbrechen.

Syntax

```
DISCONNECT [<Datenbankname>] <cursor-number>
```