

141.56.2.45

Datenbanksysteme II

Gliederung der Vorlesung

1. Erweiterungen von Datenbanksprachen

- 1.1. Übersicht zu Datenbank-Schnittstellen
- 1.2. Anweisungen von SQL in anderen Sprachen (ODBC)

2. Datensicherheit/Zugriffsschutz

- 2.1. Übersicht Datensicherheit /Transaktionen
- 2.2. Sicherung der semantischen Integrität
- 2.3. Sicherung der operationalen Integrität
- 2.4. Sicherung der physischen Integrität
- 2.5. Zugriffsschutz/Datenschutz

3. Logische Datenmodelle

- 3.1. Logische und physische Datenorganisation / Überblick über Datenmodelle
- 3.2. Klassische logische Datenmodelle
- 3.3. Objektorientiertes Datenmodell
- 3.4. DBMS-spezifische Erweiterungen vom Standard-SQL (Oracle)
- 3.5. Objektrelationale Erweiterungen im RDM (Oracle)

4. Physische Datenorganisation

- 4.1. Übersicht/Abgrenzung
- 4.2. Zugriffspfade außerhalb der Datensätze/Indexierung
- 4.3. Zugriffspfade innerhalb der Datensätze/Verkettung

5. Projektierung von relationalen Datenbanksystemen

(Betriebliche Datenmodellierung und Datenbankanwendungen)

- 5.1. Lebenszyklus von Datenbankanwendungen
- 5.2. Überblick über den Entwurfsprozess von DBS
- 5.3. Phasen des Entwurfsprozesses
 - 5.3.1. Anforderungsanalyse und -spezifikation
 - 5.3.2. Konzeptueller Entwurf
 - 5.3.3. Logischer Entwurf
 - 5.3.4. Implementierungsentwurf
 - 5.3.5. Physischer Entwurf
 - 5.3.6. Nutzung/Wartung

6. Überblick zu aktuellen Entwicklungen auf dem Gebiet DBS

1. Aufgabenkomplex Praktikum SQL - MS SQL - Grundlagen

1 Informationen zu Ihrer Datenbank ermitteln

Die Vorgehensweise zur Lösung der folgenden Aufgaben ist Ihnen frei gestellt, sie sollte jedoch so effizient wie möglich sein und die Integrität der Daten bewahren!

Überprüfen Sie daher nach jeder Aufgabe Ihre Ergebnisse, ggf. dokumentieren Sie diese.

Informieren Sie sich über die Metadaten und Daten Ihrer Tabellen **Mitarbeiter**, **Projekt**, **Zuordnung**! Vergleichen Sie Ihre Tabelleninhalte!

101 / 102 → darf nicht drin sein

Tabelle Mitarbeiter (ohne die von Ihnen eingefügten Datensätze)

MitID	Nachname	Vorname	Ort	Gebdat	Beruf	Telnr
103	Fuchs	Reinecke	Dresden	1993-07-19	Dipl.-Ing.	03501/002974
104	Elster	Diebische	Dresden	1989-01-03	Dipl.-Math.	03521/3333
105	Uhu	Uhu	Ast	1983-10-03	Dipl.-Ing.	03521/3419
106	Rabe	Weiser	Schule	1976-12-13	Dipl.-Inform.	0351/993586
107	Silie	Peter	Garten	1969-09-20	Dipl.-Inform.	0351/993386
108	Grafie	Otto	Dresden	1995-11-30	Dipl.-Math.	0351/371244
109	Nass	Anna	Dresden	1994-05-05	Dipl.-Ing.	0351/331977
110	Uhr	Klaus	Dresden	1987-05-27	Vertreter	0351/222222
111	Dorant	Theo	Radebeul	1980-05-04	Dipl.-Kaufm.	0351/1112222
112	Hase	Hoppeline	Freital	1978-10-28	Dipl.-Ing.	03501/444499
113	Nager	Panscho	Radebeul	1982-10-04	Dipl.-Math.	0351/991815
115	Mickey	Mouse	Disneyland	1970-06-27	Dipl.-Ing.	03731/11109
116	Duck	Donald	Disneyland	1975-02-28	Dipl.-Math.	03731/11118
119	Rubble	Barney	Steintal	1995-09-24	Azubi	035203/2225
121	Muppet	Kermit	Garten	1995-04-29	Dipl.-Inform.	0351/1100313
130	Simpson	Homer	Springfield	1995-11-30	Dipl.-Ing.	03528/90909
135	Feuerstein	Fred	Steintal	1990-11-11	Vertreter	0351/110055
141	Rabbit	Roger	Bau	1994-01-23	Azubi	035203/0081
145	Feuerstein	Wilma	Steintal	1995-11-30	Dipl.-Ing.	0351/110000

Tabelle Projekt (ohne die von Ihnen eingefügten Datensätze)

ProNr	ProName	ProOrt	Beschreibung	Aufwand	Leiter
31	Reportgenerator	Berlin	NULL	3	Hase
32	Statistikpaket	Dresden	NULL	2	Nager
33	Sybase-Test	Leipzig	NULL	3	Igel
34	Textgenerator	Freital	NULL	3	Fuchs
35	Fahrplanerst.	Dresden	NULL	2	Elster
36	Grafikpaket	Pirna	NULL	3	Silie
37	Fahrk.-Autom.	Chemnitz	NULL	4	Nass
38	Computerspiele	Leipzig	NULL	4	Uhu
39	Daten SAP	Dresden	NULL	0	NULL
41	Oracle-Install.	Dresden	NULL	2	Simpson
42	SAP-Customizing	Freiberg	NULL	5	Duck
43	WEB-Anwendung	Cottbus	NULL	3	Rabe

ProName ≠ TitName

→ TitID = TitName als Kennwert unterliegen müssen

Tabelle Zuordnung (ohne die von Ihnen eingefügten Datensätze)

MitID	ProNr	Istanteil	Plananteil
103	34	0,4	0,5
104	35	0,5	0,5
105	38	0,5	0,5
106	36	0,1	0,1
106	43	0,7	0,8
107	31	0,5	0,4
107	36	0,5	0,5
108	31	0,7	0,6
109	31	0,5	0,4
109	37	0,5	0,5
111	34	0,4	0,3
111	37	0,5	0,5
112	31	0,9	0,7
112	33	0,2	0,3
113	32	0,5	0,5
116	42	0,9	0,9
121	35	0,6	0,6
130	36	0,3	0,4
130	41	0,7	0,6
141	44	0,6	0,6

Falls in Ihren Tabellen – speziell in Tabelle Mitarbeiter – Daten fehlen, tragen Sie diese auf möglichst rationelle Weise wieder ein!

Vgl. dazu die Tabellen

depotN..quelleMitarbeiter, depotN..quelleZuordnung und depotN..quelleProjekt

2 Metadaten/Tabellendefinition der Tabelle Projekt sowie Daten anpassen

Hinweis: Die Eigenschaft „NULL-Werte erlaubt“ in der Tabelle Projekt bedeutet inhaltlich, dass ein NULL-Wert steht, wenn für ein neues Projekt der *Leiter* noch nicht bestimmt ist oder das Projekt fertig gestellt und der Leiter von diesem Projekt entbunden ist.

- ~~2.1~~ Fügen Sie in der Tabelle **Projekt** eine Spalte hinzu, so dass der Leiter des Projekts eindeutig identifiziert werden kann.
- ~~2.2~~ Fügen Sie anhand der bisherigen Daten in der Spalte Leiter die passenden Daten in die neue Spalte ein. Gehen Sie möglichst effizient vor.
- ~~2.3~~ Ersetzen Sie alle nicht mehr im Unternehmen arbeitenden Leiter durch noch anwesende Mitarbeiter. Nach Abschluss der Änderungen darf in der neuen Spalte kein NULL-Wert vorhanden sein, wenn bisher für das Projekt ein Leiter eingetragen war. Die bisherigen NULL-Werte in der Spalte Leiter sollen in die neue Spalte übernommen werden.
- ~~2.4~~ Löschen Sie die alte Spalte Leiter.

3 Einfügen, Ändern von Daten

- 3.1 Tragen Sie Ihre persönlichen Daten in die Tabelle **Mitarbeiter** ein. Die **MitID** sei „210“, der **Beruf** „Student“.
- 3.2 Geben Sie in der Tabelle **Projekt** ein neues Projekt „MS-SQL-Prakt“ mit **ProNr 46** ein, die **ProOrt** und **Beschreibung** seien frei wählbar, der Aufwand sei eine Zahl zwischen 2 und 4. Tragen Sie sich selbst als Leiter ein.
- 3.3 Ordnen Sie sich das Projekt (**ProNr 46**) in der Tabelle Zuordnung mit einem **Istanteil < 1.0** und einem **Plananteil** von mindestens 0.65 zu.
- 3.4 Ordnen Sie sich ein beliebiges weiteres Projekt zu!
Die Summe Ihrer **Istanteile** und **Plananteile** soll 1.0 ergeben.
- 3.5 Erteilen Sie Herrn Rabe ein weiteres Projekt, so dass dessen **Plananteil-Summe** ebenfalls 1.0 beträgt. **106**
- 3.6 Tragen Sie ein weiteres Projekt (z. B. „Oracle-Prakt.“) mit der **ProNr 47** ohne Leiter mit Aufwand = 0 ein.

4 Stored procedure mit Cursor

Erstellen Sie eine Prozedur, die folgende Aufgaben erfüllt. In der Prozedur ist genau ein Cursor zu verwenden.

→ Prozedur

- a) Zu einer der Prozedur übergebenen **Projektnummer** sollen der Projektname und der benötigte **Aufwand** angezeigt werden.
Des Weiteren sollen **MitID**, **Nachname** und **Vorname** des Projektleiters angezeigt werden.
(Ausgabe soll durch ein SELECT-Statement erfolgen)
- b) → Cursor
Ergänzen Sie die, in a) erstellte Prozedur:
Im 2. Ausgabeteil sollen die dem Projekt zugeordneten Mitarbeiter mit deren **MitID**, **Nachname**, **Vorname**, **Beruf**, **Plananteil**, **Istanteil** sowie deren Abweichung (**Plananteil – Istanteil**) je Mitarbeiter ausgewiesen werden.

Sollte dem Projekt kein Mitarbeiter zugeteilt sein, ist eine kurze Meldung auszugeben.
Alle Ausgaben sollen durch eine PRINT-Anweisung im Meldungsfenster erfolgen.

Ausgaben als Text erfolgen in der Form

```
print ' text ' + @char_variable                                bzw.  
print ' text ' + CONVERT( CHAR(n), @zahl_variable )
```

mit n - hinreichend große Zahl für die Konvertierung in eine Zeichenkette

5 Abfragen

Formulieren Sie die folgenden Abfragen, soweit möglich, sowohl als implizite Joins (WHERE-Klausel), als auch als explizite Joins (Join on-Klausel). Unterabfragen sollen nur dann genutzt werden, wenn ein Join direkt über die Tabellen nicht möglich ist.

Setzen Sie nur die zwingend erforderlichen Tabellen im SQL-Statement ein!

- 5.1 Zeigen sie für alle Projektleiter deren **Nachnamen**, **Vorname**, **Ort** sowie die zu leitende/n Projekt/n mit den jeweiligen Projektnamen an.
- 5.2 Ermitteln Sie für die an Projekten arbeitenden Mitarbeiter die Mitarbeiterdaten, an welchen Projekt (NUR **ProNr**) sie arbeiten und welchen **Istanteil** sie in dem jeweiligen Projekt haben.
- 5.3 Zeigen Sie die Projektdaten mit einem **Aufwand >= 3** sowie die Daten der zugehörigen Projektleiter (**Nachname**, **Vorname**, **Ort**) und deren **Istanteil** in dem Projekt an.
- 5.4 Listen Sie für **ALLE Projekte** **ProNr**, **Proname** und die **Summe der Istanteile** auf, auch für die Projekte, die z. Z. nicht bearbeitet werden. Erstellen Sie diese Abfrage ...
 - a) als **JOIN** (beachten Sie die Wahl des richtigen JOINs) sowie
 - b) als **UNION** (Für eine „leere“ Spalte muss ein (konstanter) DUMMY-Wert mit passendem Datentyp im SELECT-Statement gesetzt werden).
- 5.5 Ermitteln Sie für **ALLE Mitarbeiter** die Reserven als Differenz zwischen 1 und ihrer Gesamtplananteile. (Hinweis: Nutzen Sie die Erfahrung aus Aufgabe 5.4.)

2. Aufgabenkomplex SQL-Praktikum

Datenbankzugriff mittels ODBC in C

Es ist ein C-Programm zu schreiben, in dem über die ODBC-Schnittstelle auf eine MS-SQL-Datenbank zugegriffen wird. Das C-Programm soll dabei Abfragen auf die Tabellen **Mitarbeiter**, **Zuordnung** und **Projekt** ausführen.

Die nachfolgenden Aufgaben sollen als Funktionen aus der main-Routine heraus aufgerufen werden. Die Funktionalität des MSSQLUtil-Moduls (MSSQLUtil.h / MSSQLUtil.c) ist für die Lösung der Aufgaben nutzbar.

Der Programmaufruf kann ggf. mit den Parametern MSSQL-Login, MSSQL-Passwort erfolgen.
Der Programmaufruf kann ggf. mit den Parametern MSSQL-Login, MSSQL-Passwort erfolgen.

```
int main(int argc, char *argv[])
```

1. Teil-Aufgabe „Beruf“

Anzeige aller verschiedenen **Berufe** aus der Tabelle **Mitarbeiter** für Auswahlzwecke.
Auswahl eines **Berufes**.

weglassen

2. Teil-Aufgabe „Mitarbeiter“

Anzeige aller Mitarbeiter mit **MitID**, **Nachnamen** und **Vornamen** für den ausgewählten Beruf.
Auswahl einer Mitarbeiternummer. Sollte der **Beruf** nicht vorhanden sein, ist eine Meldung auszugeben.

3. Teil-Aufgabe „Projekte“

Es ist sicherzustellen, dass **nur Mitarbeiter** aus der in Teil-Aufgabe 2 erstellten Liste ausgewählt werden können.

Anzeige folgender Daten für die ausgewählte Mitarbeiternummer: **Nachname**, **Vorname**, **Ort**, **Beruf**, **Gebdat**, **Telnr.**

Des Weiteren sollen alle von ihm bearbeiteten Projekte mit **ProNR**, **ProName**, **Istanteil**, **Plananteil** aufgelistet werden.

Sollte der Mitarbeiter kein Projekt haben, ist dieses anzugeben.

ein Funktion

„Leer“eingabe (ENTER) soll in jeder Funktion den Übergang zur übergeordneten, aufrufenden Funktion sein bzw. das Ende des Programmes bewirken.

Bei fehlerhaften Eingaben soll die aktuelle Auswahlliste wiederholt werden.

zurück

Hinweise:

Als Vorlage wird das Programm MSSQLKunde.c bereitgestellt. Dieses Beispielprogramm zum Zugriff auf die Kunden-DB ist zu finden unter:
\\141.56.20.61\prakt (meist als Netzlaufwerk I oder P vorab eingebunden)

\graefe\DBSII_ODBC\Windows\KundeDB

Die Datei Kunde.sln umfasst ein vollständig lauffähiges MS-Visual Studio C - Projekt, in dessen Unterverzeichnis \Kunde2 die notwendigen Quellcode- und Header-Files gespeichert sind.

Kopieren Sie dieses Beispiel in Ihr eigenes Verzeichnis – BITTE NICHT IM NETZLAUFWERK ÖFFNEN!

Voraussetzung für die Weiterarbeit:

Analysieren und verstehen Sie das Beispiel-Projekt „Kunde“ !!!

Vorgehen:

Variante 1)

Sie setzen unter MS-Visual Studio eine neue C-Programm-Konsolenanwendung für das Projekt „Mitarbeiter“ auf, in der Sie die Aufgabe bearbeiten.

Kopieren Sie die Dateien MSSQLUtil.h und MSSQLUtil.c in den entsprechenden Quelltext-Ordner! Schreiben Sie das C-Programm gemäß Aufgabenstellung.

Variante 2)

Loggen Sie sich auf der ilux150 ein, denn dort sind die entsprechenden ODBC-Treiber installiert sind und arbeiten Sie unter Linux. Das entsprechende Projektbeispiel für Linux finden Sie unter \graefe\DBS_II_ODBC\Linux\KundeDB

Kopieren Sie dieses in Ihr Linux-Homeverzeichnis.

Im Linux- Homeverzeichnis müssen die Dateien **.odbc.ini** und **.freetds.conf** abgelegt sein.

Dort ist die zu verwendende Datenbank einzutragen (Verkauf oder Ihre eigene DB)

.odbc.ini

```
[MSSQLSERVER]
Driver      = FreeTDS
Description   = ODBC connection via FreeTDS
Trace        = No
Servername = idb45
Database    = Verkauf
Port         = 1433
ReadOnly     = No
```

.freetds.conf

```
# Server LV Datenbanken
[idb45]
host = 141.56.2.45
port = 1433
TDS_Version = 8.0
client charset = UTF-8
ReadOnly = No
```

Legen Sie in Ihrem Homeverzeichnis ein Unterverzeichnis „Mitarbeiter“ an und kopieren Sie das **Makefile** und die Dateien **MSSQLUtil.h** und **MSSQLUtil.c** da hinein. Programmieren Sie die Übungsaufgabe in einer neuen Quelltextdatei **Mitarbeiter.c**.

Der Befehl „**make all**“ startet den C-Compiler und Linker. Es entsteht das Programm *Example*. Dieses wird mit der Anweisung **./Example** gestartet.

3. Aufgabenkomplex - Semantische Integrität

Teil I: Benutzerdefinierte Funktionen

1 Skalarwertfunktion

Schreiben Sie eine Skalarwertfunktion *AlterErmitteln*, die das richtige Alter zu einem übergebenen Datum berechnet.

Testen Sie Ihre Funktion, indem Sie sich alle Daten zu Ihren Mitarbeitern sowie zusätzlich das Alter ausgeben lassen.

2 Tabellenwertfunktion

Entwickeln Sie eine Tabellenwertfunktion, die Ihnen eine Tabelle mit den Spalten *MitID* und *Auslastung* zurückgibt. Auslastung soll hierbei die Summe der Istanteile für den Mitarbeiter sein. Es sollen nur die Mitarbeiter ausgegeben werden, die in ihrer Auslastung über einem übergebenen Schwellwert liegen.

Testen Sie die Funktion!

Teil II: Sicherung der semantischen Integrität – Deklarative Lösung

Überprüfen Sie stets Ihre Lösungen anhand geeigneter Beispiele!

1 DEFAULTS

1.1 In der Tabelle *Mitarbeiter* gelten die Annahmen,

- d) dass die meisten Mitarbeiter ihren Wohnort (Ort) in Dresden haben,
- e) dass der *Beruf* bei den meisten "Dipl.-Ing." ist.

Damit können diese Eingaben bei neuen Mitarbeitern entfallen, wenn diese Angaben zutreffend sind.

2 CHECK

2.1 Ergänzen Sie die Tabellendefinition *Mitarbeiter*:

- f) Das Alter muss zwischen einschließlich 18 und einschließlich 60 liegen. Nutzen Sie Ihre benutzerdefinierte Funktion *AlterErmitteln*
- g) Jede der 3 Stellen der *MitID* muss mit einer Ziffer belegt werden.

2.2 In die Tabelle *Zuordnung* dürfen nur Datensätze eingefügt werden, in denen der Istanteil $\leq 1,0$ ist.

Teil III: Sicherung der referentiellen Integrität – Deklarative Lösung

3 Sicherung der referentiellen Integrität auf deklarativem Wege

- 3.1 Sichern Sie auf deklarativem Weg die referentielle Integrität zwischen den Tabellen **Mitarbeiter** und **Projekt**. Überprüfen und **dokumentieren** Sie die folgenden Versuche:
- h) Geben Sie in die Tabellen **Mitarbeiter** und **Projekt** je einen neuen Datensatz mit gleicher **MitID** (in Projekt ist dies die Leiternummer) ein!
Versuchen Sie anschließend in **Mitarbeiter** die **MitID** in eine andere, bisher nicht vorhandene **MitID** zu ändern.
 - i) Versuchen Sie danach diesen neuen Datensatz in **Mitarbeiter** zu löschen.
 - j) Ändern Sie diese neue Leiternummer in **Projekt** in eine andere, in der Tabelle **Mitarbeiter** vorhandene **MitID**. Wiederholen Sie das Ändern und Löschen der **MitID** (aus Aufgabe a) in **Mitarbeiter**.
 - k) Versuchen Sie, in **Projekt** einen Datensatz mit einer nicht existierenden Mitarbeiternummer einzutragen.
 - l) Versuchen Sie, in **Projekt** einen Datensatz mit Projektleiter-Nr.= NULL einzutragen

Hinweis: Die Eigenschaft „NULL-Werte erlaubt“ in der Tabelle **Projekt** bedeutet inhaltlich, dass ein NULL-Wert steht, wenn für ein neues Projekt der *Leiter* noch nicht bestimmt **ist** oder das Projekt fertig gestellt und der Leiter von diesem Projekt entbunden ist.

Teil IV: Sicherung der referentiellen bzw. semantischen Integrität – Prozedurale Lösung

Allgemeine Bemerkungen zur Aufgabenstellung für die TRIGGER

- Die Trigger sollen stets für das Einfügen, Löschen, Ändern mehrerer Datensätze funktionieren.
- Zeigen Sie die eingefügten, gelöschten oder veränderten Datensätze und bei Fehlern die fehlerhaften Datensätze an!
- In der Testphase sollten Sie mit den Kommandos "**begin transaction | rollback transaction**" den Urzustand Ihrer Tabellen erhalten.
- Mit **sp_depends [Tabellenname] | Triggername]** können Sie die Abhängigkeiten prüfen.
- Überprüfen Sie stets Ihre Ergebnisse!
- Beachten Sie, dass jeweils nur ein Insert-, Update- und Delete-Trigger je Tabelle existieren kann. Erweitern Sie deshalb gegebenenfalls bereits vorhandene Trigger.

Beachten Sie im Folgenden für das Testen der Trigger alle Constraints (Bedingungen) auf einer Tabelle bzw. zwischen den Tabellen. KEINE dieser Bedingung soll gelöscht werden.

Hinweis für die Arbeit mit dem SQL Developer:

Auf Grund eines Problems in der Kommunikation von Client, JDBC-Treiber und Datenbank setzt der SQL-Developer nach jedem Befehl ein Autocommit. Um eine Transaktion zu starten, müssen Sie folgende Vorgehensweise nutzen:

```
/*sqldev:stmt*/begin transaction;  
select / update etc.  
/*sqldev:stmt*/rollback / commit;
```

4 Trigger zur Sicherung der semantischen Integrität

- 4.1** Sichern Sie ab, dass bei entsprechenden Datenmanipulationen die Summe der geplanten Projekttätigkeiten (**Plananteil**) für einen Mitarbeiter 1.0 nicht überschreitet.

- Testen Sie den Trigger indem Sie
- m)** für alle Mitarbeiter des Projektes 31 den Plananteil um 0.3 erhöhen.
(... set Plananteil = **Plananteil** + 0.3)
 - n)** Den Plananteil für den Mitarbeiter 105 um 0.4 erhöhen.

- 4.2 Stellen Sie sicher, dass mit dem Löschen des letzten Datensatzes zu ein und demselben Projekt aus der Tabelle **Zuordnung** das Projekt tatsächlich als erledigt gekennzeichnet wird. Beachten Sie, dass es auch möglich ist, Projektnummern zu ändern.

Testen Sie den Trigger indem Sie

- o) nacheinander die Mitarbeiter zum Projekt 36 löschen.
- p) alle Mitarbeiter des Projektes 37 in das Projekt 35 versetzen.

5 Trigger zur Protokollierung von Datenänderungen und zum Kaskadierenden Löschen

Jede Änderung des **Berufes** - und nur die des **Berufes** - eines Mitarbeiters soll mit Hilfe eines Triggers mitprotokolliert werden. Es ist zunächst eine Tabelle aufzubauen, in der bei jeder UPDATE-Operation je ein Datensatz mit den Informationen zur Mitarbeiternummer, der Eintragung im Feld **Beruf** vor und nach der Änderung generiert wird.

Zusätzlich soll der Nutzer sowie Datum und Uhrzeit der Änderung erfasst werden.
(Nutzen Sie dazu die Funktionen `user_name()` und `getdate()`).

- 5.1 Richten Sie die Protokolltabelle in folgender Form ein: **Tabelle Bprotokoll**

MitID	Nutzer	Zeit	Beruf_alt	Beruf_neu
char(3)	char(16)	datetime	char(15)	char(15)

- 5.2 Erstellen Sie einen Trigger, der jede Änderung (UPDATE) des **Berufes** eines / mehrerer* **Mitarbeiter** in der Tabelle **Bprotokoll** protokolliert – *d. h. für die gleichzeitige Änderung von mehreren Datensätzen.

- 5.3 Führen Sie anschließend eine Änderung des **Berufes** bei ausgewählten Mitarbeitern durch. Überprüfen Sie die Eintragungen in der **Protokolltabelle**!

- 5.4 Welche(s) Attribut(e) würden einen geeigneten Primärschlüssel bilden.

Kaskadierendes Löschen

- 5.5 Erstellen Sie einen Trigger, der mit dem Löschen von Mitarbeitern (Tabelle **Mitarbeiter**) deren Datensätze aus der Tabelle **Bprotokoll** löscht.
Testen Sie den Trigger, wählen Sie dazu geeignete Daten.

Trigger zur Protokollierung von Datenänderungen - (Fortsetzung)

- 5.6 Fügen Sie in der Tabelle **Bprotokoll** eine Spalte hinzu, die als **Datensatzzähler** im Sinne eines **AUTOINCREMENT** dient, der beim Einfügen von Datensätzen automatisiert zählt und gleichzeitig Primärschlüssel ist.
- Überprüfen Sie Ihre Tabelle **Bprotokoll** bezüglich der Daten und Metadaten.
- 5.7 Passen Sie ggf. den Trigger aus 6.2 an und testen Sie erneut.
- 5.8 Legen Sie eine weitere Tabelle **Bprotokoll2** (analog zu **Bprotokoll**) mit einer zusätzlichen Spalte **DSNo** vom Typ **int** mit **Primärschlüsseleigenschaft** an. In dieser Spalte soll eine im folgenden Trigger eigenständig berechneter Datensatzzählung erfolgen.
- 5.9 Erstellen Sie einen Trigger, der wiederum automatisch jede Änderung (UPDATE) eines **Berufes** in der Tabelle **Mitarbeiter** - zur **Vereinfachung nur Änderung eines einzelnen Datensatzes** - in der Tabelle **Bprotokoll2** protokolliert und dabei die Spalte **DSNo** bei jedem neuen Datensatz berechnet. (KEIN Cursor erforderlich).
Testen Sie diesen Trigger!
- 5.10 Ändern Sie Ihren Trigger 6.10 so, dass auch für mehrere gleichzeitige Änderungen eines **Berufes** in der Tabelle **Mitarbeiter** in **Bprotokoll2** protokolliert und die Spalte **DSNo** berechnet wird.
Testen Sie diesen Trigger!

4. Aufgabenkomplex für das SQL–Praktikum – ACID-Prinzip, Zugriffsrechte

Hinweis für die Arbeit mit dem SQL Developer:

Auf Grund eines Problems in der Kommunikation von Client, JDBC-Treiber und Datenbank setzt der SQL-Developer nach jedem Befehl ein Autocommit. Um eine Transaktion zu starten müssen Sie folgende Vorgehensweise nutzen:

```
/*sqldev:stmt*/begin transaction;  
select / update etc.  
/*sqldev:stmt*/rollback / commit;
```

1 Aufgaben zum Transaktionsprinzip – nur eigene DB betreffend

1. Beginnen Sie eine Transaktion, in der Sie den Inhalt der Tabelle **Zuordnung** löschen und danach lesen.
2. Setzen Sie die Transaktion wieder zurück und lesen Sie **Mitarbeiter** erneut.
3. Fassen Sie in Stichpunkten Ihre Erkenntnisse zum ACID-Prinzip zusammen

2 Aufgaben zu Zugriffsrechten

Lösen Sie die folgenden Aufgaben wechselseitig mit einem Partner in Form eines Rollenspiels.

Protokollieren Sie Ihre Ergebnisse und Erkenntnisse („Sie“ als DBO → user_a, „Partner“ als Fremdnutzer → user_b).

Nutzen Sie folgende System-Prozeduren, um Ihre Aktivitäten auf der Datenbank zu beobachten:

sp_who	Gibt Informationen über alle aktuellen Server-Benutzer und -Prozesse bzw. über einen bestimmten Benutzer oder Prozess aus
sp_lock	Gibt Informationen über Prozesse aus, die derzeit Sperren verursachen.
sp_helpuser	Liefert Informationen über einen bestimmten Benutzer, eine Gruppe, ein Alias oder über alle Benutzer in der aktuellen Datenbank.
sp_helpprotect	Anzeige der Rechte eines Benutzers auf ein Objekt

1. User B: Wechsel in die Datenbank von User A
2. Lassen Sie Ihren Partner als Anwender Ihrer Datenbank zu. Das Default-Schema soll „extern“ heißen.
3. Legen Sie das Schema „extern“ an und autorisieren Sie Ihren Partner dafür.
4. User B: Wechsel in die Datenbank von User A

5. Ihr Partner soll versuchen, den Inhalt Ihrer Tabelle **Mitarbeiter** zu lesen.
6. Erteilen Sie Ihrem Partner die Leserechte für Ihre Tabelle **Mitarbeiter**.
7. Ihr Partner soll erneut versuchen, den Inhalt Ihrer Tabelle **Mitarbeiter** zu lesen.
8. Informieren Sie sich über zugelassene Nutzer zu Ihrer Datenbank und über vergebene Zugriffsrechte.
9. Fordern Sie Ihren Partner auf, eine Transaktion zu beginnen und als erste Operation ein Lesen der Tabelle **Mitarbeiter** mit Schreibabsicht (Benutzung von **holdlock**) durchzuführen.
10. Versuchen Sie selbst, einen Wert in **Mitarbeiter** zu ändern.
11. Prüfen Sie, ob jemand Ihre Tabelle **Mitarbeiter** gesperrt (locks) hat und wer es war (Über „Neue Abfrage“ eine zweite Verbindung zur Datenbank öffnen).
12. Lassen Sie Ihren Partner die Transaktion abschließen. Versuchen Sie erneut, einen Wert in **Mitarbeiter** zu ändern.
13. Erteilen Sie Ihrem Partner das Recht, in Ihrer Datenbank eine Tabelle anzulegen.
14. Ihr Partner soll eine Tabelle **Mitarbeiter** (analog DBS I – Aufg. 1.1) anlegen und (nur) einen Datensatz einfügen.
15. Sehen Sie sich in Ihrer DB die Tabelle **Mitarbeiter** Ihres Partner an.
16. Überprüfen Sie die Objekte in Ihrer Datenbank.
17. Ihr Partner soll die von ihm angelegte Tabelle **Mitarbeiter** wieder löschen.
Tauschen Sie die Rollen und beginnen Sie danach wieder bei 1.
18. Beide !!!
Entziehen Sie Ihrem Partner die Zugriffsrechte wieder, und löschen Sie unbedingt den Fremd-User sowie das angelegte Schema aus Ihrer Datenbank!