

3 SQL

3.1	Übersicht.....	3
3.1.a	Merkmale	4
3.1.b	Komponenten.....	5
3.2	DML.....	6
3.2.a	Queries	6
3.2.a.1	Select	7
3.2.a.2	Where	9
3.2.a.3	Between	12
3.2.a.4	In	12
3.2.a.5	Like.....	13
3.2.a.6	Übersicht DML.....	13
3.2.a.7	Sortierung	14
3.2.a.8	Arithmetische Operationen.....	14
3.2.a.9	Aggregatfunktionen.....	15
3.2.a.10	Gruppierung.....	16
3.2.b	Tabellen verknüpfen	17
3.2.b.1	Union	18
3.2.b.2	Joins.....	18
3.2.b.3	Auto Join	21
3.2.b.4	Subqueries	22
3.2.c	Insert	25
3.2.d	Update.....	26
3.2.e	Delete.....	26
3.3	DDL.....	27
3.3.a	Create Table.....	27
3.3.a.1	Large Objects	28
3.3.a.2	Constraints.....	29
3.3.a.3	Daten-Integrität	30
3.3.a.4	Index.....	31
3.3.b	View.....	33
3.4	DCL.....	34
3.4.a	Transaction.....	34
3.4.a.1	Commit / Rollback	35
3.4.a.2	Lock-Mechanismen	36
3.4.a.3	Isolation Levels	37
3.4.a.4	2 Konzepte.....	38
3.4.a.5	Transaction Log.....	39
3.4.a.6	Recovery.....	40
3.4.b	Datenschutz.....	41
3.5	Procedurale Elemente.....	45
3.5.a	Stored Procedure.....	45
3.5.b	Cursor.....	47
3.5.c	Trigger	48
3.6	Übungen.....	49
3.6.a	Team, SQL Abfragen.....	49
3.6.b	Firma, SQL Abfragen	56
3.6.c	DDL und DML	62
3.6.d	Constraints	63
3.6.e	Index, View.....	64
3.6.f	Transaktion	65
3.6.g	Benutzerrechte, Sicherheitsmechanismen.....	66
3.6.h	Stored Procedures, Triggers.....	67

3.7	Übungen mit MS SQL Server	68
3.7.a	So verwenden Sie den SQL Server Query Analyzer.....	69
3.7.b	So erstellen Sie eine Datenbank mit SQL	71
3.7.c	So benutzen Sie Constraints.....	72
3.7.d	So verwenden Sie die IDENTITY – Eigenschaft.....	73
3.7.e	So verwenden Sie die Funktion NEWID() und den Datentyp uniqueidentifier	73
3.7.f	So erstellen Sie ein ERM der Northwind – Datenbank.....	74
3.7.g	So erstellen Sie eine View	75
3.7.h	So funktioniert eine Transaktion.....	76
3.7.i	So erstellen Sie eine Stored Procedure.....	78
3.7.j	So erstellen Sie einen Trigger	79

3.1 Übersicht

Höhere Fachschule Uster
Bildungszentrum Uster

SQL

- **Industrie Standard**
für alle rel. DBMS
mit allen OS
auf allen Plattformen
- **Geschichte**

1970	Codd: Relationen Modell
197x	IBM San Jose Research Lab -> SEQUEL
1977	IBM, erster Prototyp eines rel. DBMS
1979	ORACLE, erstes rel. DBMS mit SQL auf dem Markt
1984	IBM, DB2 wird verfügbar
1989	SQL-89 wird ANSI-Standard
1992	SQL2, SQL-92
1999	SQL3, OO-Features, Multimedia



Standard

- Auf dem Markt sind viele versch. Rel. DBS auf unterschiedlichsten Betriebssystemen, die auf allerlei HW laufen.
- Aber alle ernst zu nehmenden Produkte haben eines gemeinsam -> SQL
- Wer SQL kennt, kann mit den Daten auf der Datenbank umgehen, unabhängig vom DBS-Produkt, Betriebssystem, Plattform oder Hardware.
- SQL ist Standard auf allen Plattformen: PC, Mainframe, Client-Server
- Es gibt keine andere DB-Sprache von derartiger Wichtigkeit und ähnlichem Verbreitungsgrad.
- SQL ist möglicherweise die meistbenutzte Programmiersprache der Welt.

Geschichte

- Um 1970 war die DB-Technik durch Hierarchische- und Netzwerk-DB Systeme geprägt, die der Forderung nach Datenunabhängigkeit der Applikationen, Flexibilität der Datenstrukturen und Benutzung auch durch Endbenutzer in keiner Art und Weise gerecht wurde.
- Codd, Engländer, Mathematiker, 30 Jahre bei IBM -> San Jose
- Codd konnte mit seiner Arbeit theoretisch nachweisen, dass das rel. DB-Modell genau diese Mängel der vorhandenen Systeme beseitigen konnte. Es entstanden eine Vielzahl von Forschungsarbeiten, die zum Ziel hatten, die theoretische Vorgabe in ein kommerzielles DB-System umzusetzen.
- SQL-89 wurde zum ersten Standard, der auf dem kleinsten gemeinsamen Nenner der verschiedenen Hersteller beruhte und daher recht verwässert war und darum keine grosse Bedeutung erlangte.
- Mit SQL-92 gelang der Durchbruch. Er war 5x grösser als der 89-Standard, was zu einem grossen Implementations-Effort der Hersteller führte, um 92-compliant zu sein.
- SQL3 (SQL99) umfasst mehr als 2000 Seiten. Diese Zahl macht deutlich, dass eine Qualität des relationalen Datenmodells verlorengeht: die Einfachheit. Die Datenstrukturen werden komplexer und die Regeln ihrer Anwendung auch. Eine wesentliche Konsequenz wird sein, dass nicht mehr jeder normale Datenbankanwender verstehen kann, wie der Datenbestand organisiert ist. Und der Designer eines DBS hat mit viel komplexeren Entwurfsentscheidungen zu kämpfen. -> OO-Technik, Stored Procedures, Triggers, Multimedia
- Jede Firma hat ihre eigene SQL-Erweiterungen -> Portierungen sind schwierig


3.1.a Merkmale

Höhere Fachschule Uster

Bildungszentrum Uster

Merkmale

- Flexible und mächtige DB-Sprache
- Basiert auf relationaler Algebra
- Rel. leicht zu lernen
- Was und nicht Wie
- keine Oberfläche
- Bearbeitet Daten mengenweise



DB-Sprache

strukturiert, nicht prozedural, geradlinig und einfach zu verwenden, gesprochenes Englisch mit ganzen Sätzen

Rel. Algebra

SQL basiert auf relationaler Algebra

Leicht lernen

dankbare Pr.Sprache, da man schnell zu einem Erfolgserlebnis kommt
wenige Keywords (reservierte Wörter)

Was und nicht Wie

deskriptive Sprache, nur das Was muss beschrieben sein, das Wie wird durch die DB selber erledigt, sucht sich selber den besten Lösungsalgorithmus mit Hilfe von Optimizern

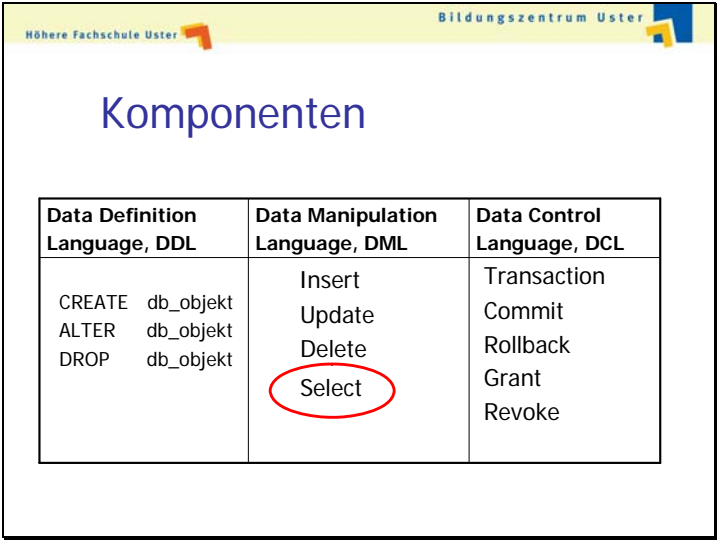
Keine Oberfläche

SQL bietet keine Möglichkeit um eine Oberfläche zu programmieren, wenn ich das aber trotzdem will, muss ich eine andere Pr.Sprache nehmen, eine Oberfläche programmieren, einen SQL-Aufruf machen, und das Result Set des DBS wieder mit der andern Pr.Sprache auf die Oberfläche bringen -> ziemlich aufwendig -> wird in diesem Skript nicht gelehrt, in Access ist die Oberfläche integriert

Mengenweise

SQL bezieht sich auf Tabellen, mit einem Befehl kann man ganze Tabellen verarbeiten, in andern Pr.Sprachen muss man, um das Gleiche zu erreichen, durch alle Datensätze iterieren, und innerhalb eines Datensatzes muss man durch alle Attribute iterieren -> 2 geschachtelte Loops

3.1.b Komponenten



Data Definition Language, DDL	Data Manipulation Language, DML	Data Control Language, DCL
CREATE db_objekt ALTER db_objekt DROP db_objekt	Insert Update Delete Select	Transaction Commit Rollback Grant Revoke

SQL-Komponenten

Wie jede andere PL hat auch SQL ihre sogenannten Key-Words. Man kann sie in 3 Gruppen aufteilen:

DDL Die DDL stellt alles zur Verfügung um eine DB zu definieren. Mit Create Table definiert man eine Tabelle mit all ihren Eigenschaften, mit Alter Table verändert man eine bestehende Tabelle und mit Drop table kann man eine nicht mehr benötigte Tabelle wieder löschen. Alle andern DB-Objekte, wie Views, Indices, STP, etc. lassen sich genauso mit Create erzeugen.

DML Die DML ist ein sehr mächtiges Werkzeug um Daten in Tabellen einzufügen, zu verändern, abzufragen und wieder zu löschen. Insbesondere der Select-Befehl ist sehr mächtig und vom Select leitet sich auch der Name SQL ab. Abfrage aus `Strukturierte Abfrage Sprache` bezieht sich auf das Select. Wir werden im Rahmen dieses Kurses darum auch vor allem mit diesem Select üben.

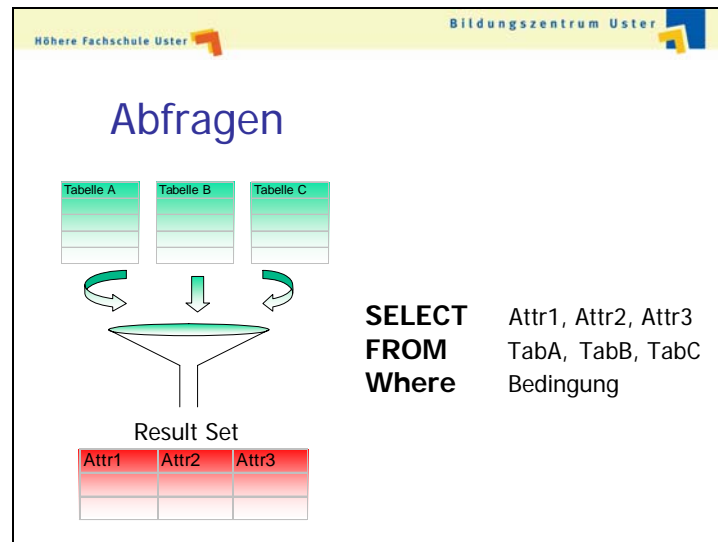
DCL Die DCL stellt alles zur Verfügung, was es braucht, um eine DB vor unerwünschten Einflüssen zu schützen. Eine DB kann auf vielerlei Weise beschädigt werden. Wenn Sie die Werkzeuge der DCL korrekt einsetzen, können Sie viele dieser Probleme vermeiden.

Generell

SELECT ist der umfangreichste und am häufigsten verwendete Befehl. Dieses Skript beginnt daher mit der Erklärung von SELECT auch darum, weil SELECT am schnellsten zu einem Erfolgserlebnis führt.

3.2 DML

3.2.a Queries



Idee

Eine Abfrage ist eine Forderung an eine Datenbank: „Bitte schicke mir Daten“.

Abfragen - Queries in English - haben die allgemeine Form:

Select	Auswahlliste der Attribute
From	Liste der beteiligten Tabellen
Where	Bedingungen

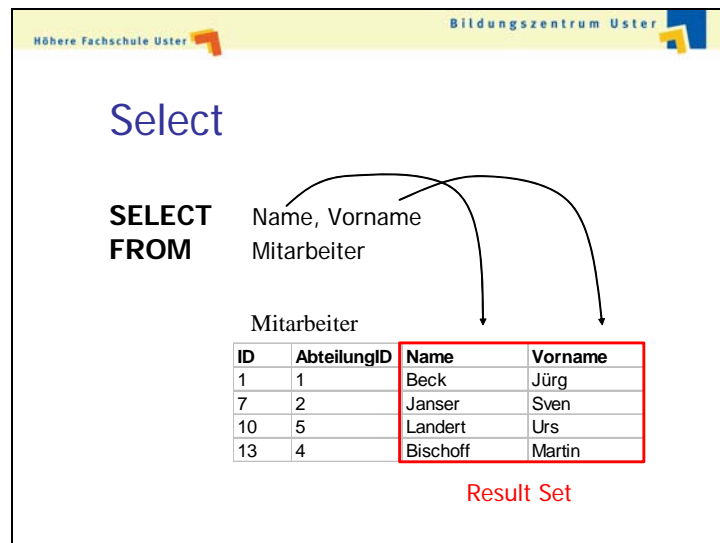
Erklärung

- Mit Select wähle ich die Attribute aus, die mich interessieren. Die Liste der Attribute wird auch als Auswahlliste bezeichnet.
- Die From – Klausel deklariert die von SELECT und WHERE angesprochenen Tabellen.
- Die Where – Klausel enthält Bedingungen (Einschränkungen), dies ist jeweils auch der anspruchsvolle Teil einer Abfrage.

Ergebnis

- Als Ergebnis einer Abfrage erhalte ich immer ein Result Set (Resultat Tabelle) zurück. Dies ist genau eine Tabelle mit 0, einem oder mehreren Datensätzen.
- Die Reihenfolge der Attribute im Result Set ist identisch zur Reihenfolge im Select-Statement.

3.2.a.1 Select



Beispiel

Dieses Beispiel selektiert alle Namen und Vornamen aus der Tabelle Mitarbeiter, ohne Einschränkungen.

Vergleich

Um in einer anderen Programmiersprache (z.B. C/C++) zum gleichen Ergebnis zu kommen, müssten 2 verschachtelte Loops programmiert sein.

Stärken von SQL


Man kann hier die Vorteile von SQL gegenüber anderen Programmiersprachen sehr deutlich erkennen:

- elegant, intuitiv, Klartext in Englisch
- funktioniert mengenmässig
- Spaltenposition ist unwichtig
- die Anzahl Datensätze in der Ausgangstabelle ist unwichtig
- nur das WAS wird deklariert
- Lösungsalgorithmus wird nicht benötigt

Höhere Fachschule Uster Bildungszentrum Uster

Ganze Tabelle

SELECT *
FROM Mitarbeiter



ID	AbteilungID	Name	Vorname
1	1	Beck	Jürg
7	2	Janser	Sven
10	5	Landert	Urs
13	4	Bischoff	Martin

Result Set

Idee

Der * hat hier die Bedeutung einer Wild card. Das heisst, zeige mir alle Attribute dieser Tabelle.

Die Kolonnen erscheinen in der Reihenfolge, in der sie bei der Definition der Tabelle kreiert wurden.

DISTINCT

Mit dem Keyword DISTINCT – dt. unterschiedlich – wird garantiert, dass nur unterschiedliche Datensätze im Result Set erscheinen. Mit andern Worten, duplikate Ergebniszeilen werden entfernt.

Select	Name, State	<u>Name</u>	<u>State</u>
From	Author	Hunter	CA
		McBadden	CA
		McBadden	CA
		Ringer	UT

Select	distinct Name, State	<u>Name</u>	<u>State</u>
From	Author	Hunter	CA
		McBadden	CA
		Ringer	UT

3.2.a.2 Where

Höhere Fachschule Uster Bildungszentrum Uster

Where

```
SELECT *  
FROM Mitarbeiter  
WHERE Vorname = `Sven`
```

ID	AbteilungID	Name	Vorname
1	1	Beck	Jürg
7	2	Janser	Sven
10	5	Landert	Urs
13	4	Bischoff	Martin

Result Set

Idee

Mit der WHERE-Klausel werden die Datensätze eingeschränkt.

Vergleiche

Alle üblichen Vergleichs-Operatoren sind möglich:

gleich	=
ungleich	<>
kleiner	<
grösser	>
kleiner gleich	<=
grösser gleich	>=

Schreibweise

- SQL ist **nicht case sensitiv**, man kann alles gross, alles klein oder gemischt schreiben, es macht keinen Unterschied.
- Zudem ist SQL eine **Freiformatsprache**, d.h. Es gibt keine Regeln darüber wie viele Wörter auf einer Zeile stehen dürfen. Eine Abfrage kann auf einer Zeile sein oder Sie können nur ein Wort pro Zeile schreiben.

Verknüpfungen

Logische Verknüpfungen

```

SELECT *
FROM Mitarbeiter
WHERE Vorname = `Sven`
OR AbteilungID = 4
  
```

ID	AbteilungID	Name	Vorname
1	1	Beck	Jürg
7	2	Janser	Sven
10	5	Landert	Urs
13	4	Bischoff	Martin

Result Set

- Logische Verknüpfungen von mehreren Bedingungen sind möglich mit den Operatoren: AND, OR
- Runde Klammern helfen, um die beabsichtigte Reihenfolge zu setzen. AND bindet ohne Klammern stärker als OR.

Adressierung einzelner Zellen

Einzelne Zellen

```

SELECT ID, Name
FROM Mitarbeiter
WHERE AbteilungID = 2
OR AbteilungID = 4
  
```

ID	AbteilungID	Name	Vorname
1	1	Beck	Jürg
7	2	Janser	Sven
10	5	Landert	Urs
13	4	Bischoff	Martin

Result Set

Die Adressierung von einzelnen Zellen geschieht durch Auswahl der Kolonnen mit einem SELECT-Statement und der Einschränkung der Datensätze mit der WHERE-Klausel.

Kolonnen im Result Set umbenennen

Per Default entsprechen die Kolonnenbezeichnungen im Result Set den Attributnamen aus den entsprechenden Tabellen. Manchmal ist eine andere Bezeichnungen gewünscht. Mit dem Keyword **AS** kann dies sehr einfach realisiert werden.

```

Select ID as `MitarbeiterNr`, Name as `MitarbeiterName`
From Mitarbeiter
Where AbteilungID >= 4
  
```

<u>MitarbeiterNr</u>	<u>MitarbeiterName</u>
10	Landert
13	Bischoff

Höhere Fachschule Uster
Bildungszentrum Uster

Übersicht Prädikate

- Vergleiche: = <> < <= > >=
- BETWEEN
- IN
- LIKE

Verknüpfungen mit AND, OR
Verneinung mit NOT

Wir haben die WHERE Klauseln bereits verwendet ohne deren Bedeutung zu erklären, weil die Verwendung der Klausel so intuitiv einfach ist. Die Bedingung kann beliebig einfach oder komplex sein. Mehrere Bedingungen können durch die logischen Operatoren AND, OR oder NOT miteinander verknüpfen sein. Die Bedingungen dieser WHERE Klauseln werden auch als Prädikate bezeichnet.

Weitere WHERE Klauseln: (brauchen nicht gelernt zu werden)

Ein **EXISTS** braucht man, um festzustellen, ob eine Unterabfrage eine Zeile zurückgibt. Wenn ja, gibt EXISTS true zurück, dh. die äussere Abfrage wird ausgeführt.

```
SELECT    vorname
FROM      kunde
WHERE EXISTS ( SELECT DISTINCT Kunden_ID FROM verkauf WHERE (...) )
```

Ein **MATCH** wird zu Übereinstimmungen von Datensätzen gebraucht.

```
SELECT    ...
FROM      ...
WHERE     ( @KundenID, `Artikel1`, `01-04-1997` )
MATCH     ( SELECT KundenID, Artikel, Datum FROM Tabelle )
```

Ein **UNIQUE** wird zur Feststellung benötigt, ob alle Zeilen in einer Unterabfrage eineindeutig sind.

```
SELECT    vorname
FROM      kunde
WHERE     UNIQUE ( SELECT id FROM verkauf WHERE verkauf.id = kunde.id )
➔ gibt alle Namen zurück, für die es in der Tabelle Verkauf nur einen Datensatz gibt.
```

Mit **OVERLAPS** kann man feststellen, ob sich 2 Zeitintervalle überlappen. Intervall = Startpunkt + Endpunkt, Intervall = Startpunkt + Dauer

```
SELECT    ...
FROM      ...
WHERE     ( TIME `9:00:00`, TIME `10:00:00` ) OVERLAPS ( TIME `10:15:00`,
INTERVALL `3` HOUR )
```

3.2.a.3 Between



Between - Prädikat

```

SELECT Name
FROM Nahrungsmittel
WHERE Kalorien >= 100
AND Kalorien <= 300

```

WHERE Kalorien BETWEEN 100 AND 300

Idee

Sie möchten zum Beispiel alle Nahrungsmittel auswählen, deren Energiewert zwischen 100 und 300 Kalorien liegt. Eine Methode ist die mit den Vergleichsoperatoren.

Eine andere und wahrscheinlich besser zu verstehende Methode ist die Verwendung einer BETWEEN Klausel. Wichtig zu wissen ist, dass die Grenzwerte eingeschlossen sind. Das untere Beispiel entspricht also exakt dem oberen. Ebenfalls müssen Sie als Programmierer sicherstellen, dass der erste Wert kleiner als der zweite ist, ansonsten laufen sie in einen Fehler.

Sie können das BETWEEN Prädikat für Zeichenketten, Datetime- und numerische Datentypen verwenden.

3.2.a.4 In



IN - Prädikat

```

SELECT Firma, Telefon
FROM Lieferant
WHERE Kanton IN ( `ZH`, `SG`, `AG` )

```

```

SELECT Firma
FROM Lieferant
WHERE Kanton NOT IN ( `SH`, `TG`, `GR` )

```

Idee

Mit den Prädikaten IN und NOT IN können Sie prüfen, ob der Wert eines Attributes in einer definierten Menge enthalten ist.

Beispiel

Das obige Beispiel liefert eine Liste mit Firmenname und Telefonnummer aller Lieferanten, die in Ihren Nachbarkantonen domiziliert sind.

Wissenswertes

- Die NOT IN Version dieses Prädikates funktioniert auf die gleiche Weise mit Umkehrlogik.
- Mit dem Schlüsselwort IN können Sie so – im Vergleich mit Vergleichsoperatoren - ein wenig Tipparbeit sparen, und zwar um so mehr, je mehr Elemente in der Menge enthalten sind.
- Das Schlüsselwort IN ist auch im Bereich der Unterabfragen nützlich. -> später in diesem Skript

3.2.a.5 Like

Höhere Fachschule Uster Bildungszentrum Uster

LIKE - Prädikat

```
SELECT *
FROM Freunde
WHERE Telefon LIKE `01 938 %`
```

```
SELECT Name
FROM Kanton
WHERE Kürzel LIKE `A_`
```

Idee

Mit **LIKE** kann man Zeichenketten auf gegebene Muster überprüfen.

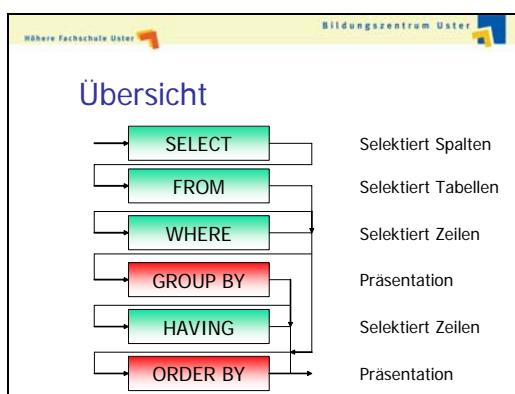
Wissenswertes

- Das % Zeichen dient als Wildcard für eine beliebige Anzahl (0 bis n) von Zeichen.
- Kennt man die Anzahl unbekannter Zeichen, benützt man den Underline als Platzhalter für genau ein Zeichen.
- Selbstverständlich kann man beide Wildcards zusammen kombinieren.
- Einige DBS verwenden * statt %.
- Access verlangt ? statt _.

Beispiel

```
Select Titel
From Bücher
Where Titel like ` %Computer%`
```

3.2.a.6 Übersicht DML



Übersicht

- Die SELECT Anweisung besteht aus 6 Komponenten, um die Spalten und Zeilen und die Sortierreihenfolge des Ergebnisses genau zu spezifizieren.
- Die ersten beiden Klauseln sind obligatorisch, die übrigen optional.
- Werden verschiedene Anweisungsteile aus der obigen Liste verwendet, muss die Reihenfolge eingehalten werden, wie sie die Übersicht darstellt.

3.2.a.7 Sortierung

Sortierung

```
SELECT zahlung, plz, name
FROM kunde
ORDER BY zahlung, plz
```

zahlung	plz	name
B	2384	Randers
B	2386	Stein
N	2386	Voss
N	2581	Berger

Idee

- Die ORDER BY Klausel sorgt für eine sortierte Ausgabe der Ergebnistabelle.
- Ohne diese Klausel ist die Reihenfolge der Datensätze in der Ergebnistabelle zufällig.
- Für die Präsentation der Resultate in einer Abfrage ist diese Klausel unverzichtbar – keinem Benutzer kann eine unsortierte Liste mit mehr als ein paar Zeilen zugemutet werden.
- Die Ausgabe wird nach den Werten der angegebenen Spalte(n) sortiert. Bei mehreren Spalten wird hierarchisch sortiert: zuerst nach der erstgenannten Spalte, bei Wertgleichheit nach der zweitgenannten Spalte etc.
- Defaultmässig wird aufsteigend sortiert, möchte man es gerne absteigend haben, setzt man einfach das Key word DESC(ending) hinten an.

3.2.a.8 Arithmetische Operationen

Die 5 Operationen	+	Addition
	-	Subtraktion
	*	Multiplikation
	/	Division
	%	Modulo

können sowohl im Select-Statement als auch in der Where-Klausel frei benutzt werden.

Beispiele

Select Name, price * 0.95 as `Sale`, price * 0.05 as `Discount`
 From Bücher

Jedes DBS bietet zusätzliche mathematische Operationen an:

- Exponential Funktionen
- Logarithmische Funktionen
- Wurzel Funktionen
- Trigonometrische Funktionen
- etc.

3.2.a.9 Aggregatfunktionen

Aggregatfunktionen	
AVG	Durchschnitt
COUNT(*)	Anzahl Datensätze
MAX	maximaler Wert
MIN	minimaler Wert
SUM	Summe aller Werte


Definition

Funktionen, die zusammenfassende Werte berechnen, werden Aggregatfunktionen genannt.

Select Avg(attribut) gibt den Durchschnitt des Attributes über alle selektierten Datensätze zurück
 Select Count(*) gibt die Anzahl selektierten Datensätze zurück
 Select Max(attribut) gibt den grössten Wert über alle selektierten Datensätze zurück
 Select Min(attribut) gibt den kleinsten Wert über alle selektierten Datensätze zurück
 Select Sum(attribut) gibt die Summe aller Werte eines Attributes über alle selektierten Datensätze zurück


- Aggregatfunktionen werden oft im Zusammenhang mit einer **Group by** Klausel gebraucht. -> weiter hinten in diesem Skript
- Aggregatfunktionen können nicht in der Where-Klausel benutzt werden -> Unterabfragen

Anzahl Zeilen	
SELECT	Count(*)
FROM	Mitarbeiter



ID	AbteilungID	Name	Vorname
1	1	Beck	Jürg
7	2	Janser	Sven
10	5	Landert	Urs
13	4	Bischoff	Martin

Maximaler Wert	
SELECT	Max(Gehalt)
FROM	Mitarbeiter
WHERE	ID > 7



ID	Name	Vorname	Gehalt
1	Beck	Jürg	5500
7	Janser	Sven	9000
10	Landert	Urs	6100
13	Bischoff	Martin	7000

Idee

Count(*) liefert die Anzahl gefundener Datensätze zurück.

Varianten

COUNT kann in 3 Varianten auftreten:

- | | | | |
|----|--------|------------------------|--|
| 1) | SELECT | Count(*) | Gibt die Anzahl Datensätze zurück |
| | FROM | Nahrungsmittel | |
| 2) | SELECT | Count(Kohlenhydrate) | Gibt die Anzahl Datensätze zurück, die in der Spalte Kohlenhydrate einen Wert haben. Diejenige Datensätze mit NULL-Werten in der Spalte Kohlenhydrate werden somit nicht mitgezählt. |
| | FROM | Nahrungsmittel | |
| 3) | SELECT | Count(DISTINCT Typ) | Gibt die Anzahl verschiedener Werte einer Spalte zurück. |
| | FROM | Nahrungsmittel | |

3.2.a.10 Gruppierung

Höhere Fachschule Uster Bildungszentrum Uster

Gruppierung

SELECT id, anzahl
FROM order

↓

id	anzahl
1	5
1	10
2	10
2	25
3	15
3	30

SELECT id, SUM(anzahl)
FROM order
GROUP BY id

↓

id	anzahl
1	15
2	35
3	45

Diagram showing the transformation of a table with multiple rows per 'id' into a grouped table where each 'id' has a single row representing the sum of 'anzahl'.

Prinzip

- Oft wollen Sie zusammengehörige Zeilen zu einer Gruppe zusammenfassen, um das Ganze besser überblicken zu können. Mit der Klausel GROUP BY können Sie genau dies tun.
- Die Gruppierung fasst mehrere Datensätze in der Basistabelle zu einem einzigen Datensatz zusammen.
- Entscheidend für die Gruppenbildung sind gleiche Werte in einer bestimmten Spalte.
- Pro Gruppe und pro Spalte gibt es nur noch einen Wert.
- Um Werte innerhalb einer Gruppe zusammenzufassen, werden Aggregatsfunktionen gebraucht (SUM, AVG, MIN, MAX, COUNT).
- Die Ausdrücke in der Auswahlliste einer Abfrage, die eine GROUP BY Klausel enthält, müssen entweder Aggregatfunktionen sein oder in der GROUP BY Liste enthalten sein, damit Zusammenfassungszeilen erzeugt werden.
- Die GROUP BY Klausel sortiert das Resultset nach dem ersten Gruppenattribut.

SQL, DML, Gruppierung

HAVING

SELECT id, SUM(anzahl)
FROM order
GROUP BY id
HAVING SUM(anzahl) > 30

id	anzahl
2	35
3	45

Diagram showing the application of the HAVING clause to filter groups based on the sum of 'anzahl'.

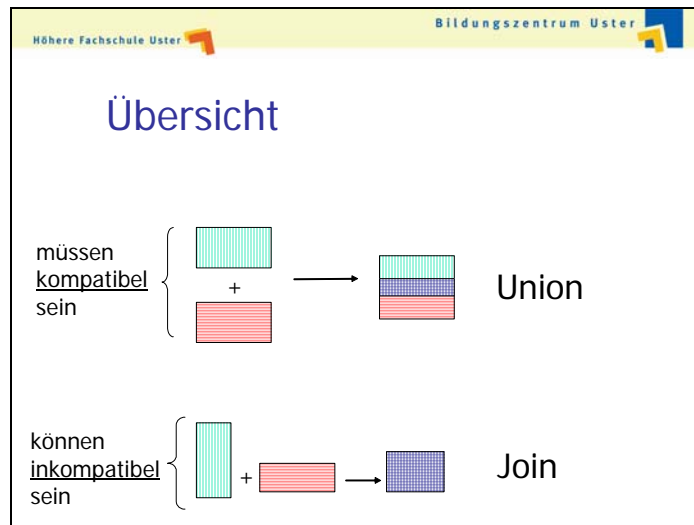
Prinzip

- Hat man das Bedürfnis die Gruppen mit einer Bedingung weiter einzuschränken, kann man dies mit einer HAVING Klausel tun.
- Die HAVING Klausel hat für die Gruppen dieselbe Bedeutung wie die WHERE Klausel für einzelne Zeilen.
- Sie können in der HAVING Klausel auf jede Spalte aus der Auswahlliste verweisen, die Klausel kann zudem Aggregatfunktionen enthalten.

Einschränkungen

- Verwenden Sie die HAVING Klausel zum Einschränken der Gruppierungen nur zusammen mit der GROUP BY Klausel. Es macht keinen Sinn, das HAVING in einem anderen Zusammenhang zu verwenden.
- In WHERE Klauseln kann nicht auf Aggregatfunktionen verwiesen werden.

3.2.b Tabellen verknüpfen



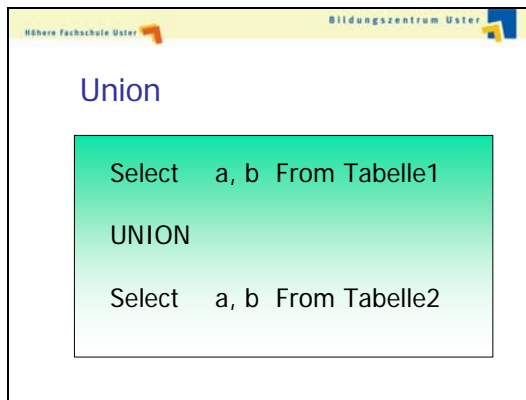
Einführung

- Zu den leistungsfähigsten Merkmalen von SQL gehört die Fähigkeit, Daten über mehrere Tabellen hinweg abzufragen und zu manipulieren. Wenn man auf diese Mechanismen nicht zurückgreifen könnte, müsste man alle für eine Anwendung erforderlichen Datenelemente in einer einzigen Tabelle speichern. Dies hätte Redundanz und längerfristig Datenanomalien zur Folge. Darum eben wird in der Regel bis in die 3. Normalform normalisiert.
- Durch diese Normalisierung werden sachlich zusammengehörige Informationen in verschiedene Tabellen zerlegt.

Idee

- Um diese sachlich zusammengehörige Information für den Benutzer wieder zusammenzufügen – z.B. für eine Liste – müssen in Abfragen mehrere Tabellen miteinander verknüpft werden.
- Bei einer Verknüpfung handelt es sich um eine Operation, mit der Sie zwei oder mehr Tabellen abfragen und auf diese Weise ein einziges Resultset erstellen können.
- SQL stellt grundsätzlich 2 verschiedene Methoden zur Verfügung:
 - Union
 - die Familie der Joins.

3.2.b.1 Union



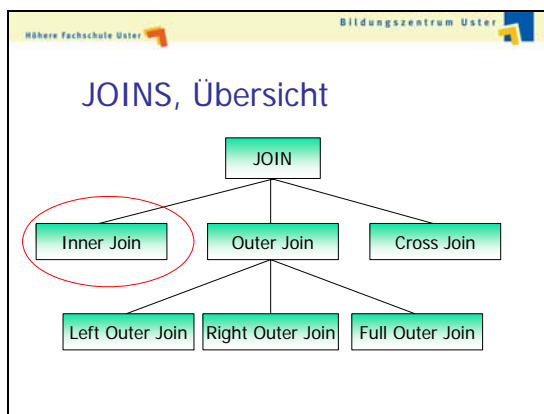
Prinzip

Der Union-Operator kombiniert 2 oder mehrere **kompatible Resultat Sets** miteinander, d.h. Anzahl Kolonnen und die Datentypen der Result Sets müssen übereinstimmen.

Identische Zeilen

Der UNION-Operator gibt alle Zeilen zurück, die in beiden Result Sets enthalten sind.
Der UNION-Operator entfernt defaultmässig alle doppelten Zeilen in der Ergebnistabelle. Wenn die doppelten Zeilen nicht entfernt werden sollen, benutzen Sie den Operator UNION ALL.

3.2.b.2 Joins



Übersicht

Die wirklich wichtigen Verknüpfungen zwischen Tabellen werden mit sogenannten Joins gemacht. Die zu verknüpfenden Tabellen brauchen hier nicht kompatibel zu sein.

Die Joins kann man in 3 Kategorien aufteilen:

- Inner Joins
- Outer Joins (können noch weiter aufgeteilt werden)
- Cross Joins

Der wichtigste und mit Abstand häufigste Join ist der INNER JOIN.

Höhere Fachschule Uster Bildungszentrum Uster

Inner Join

```

Select  k.Name, k.Id, v.ProduktID
From    Käufer k, Verkauf v
Where   k.K_ID = v.K_ID
  
```

Käufer

Name	K_ID
Baar	1
Chai	2
Corets	3
Melia	4

Verkauf

K_ID	ProduktID
1	A2
1	B3
4	D1
3	R5

Result Set

Name	K_ID	ProduktID
Baar	1	A2
Baar	1	B3
Melia	4	D1
Corets	3	R5

Prinzip

- Jeder Datensatz aus der einen Tabelle wird mit jedem anderen Datensatz aus der anderen Tabelle verglichen und geprüft, ob die Where – Klausel zutrifft.
- Schliesst alle Zeilen aus der Ergebnistabelle aus, zu der es keine Entsprechung in der jeweils anderen Tabellen gibt.
- Der Inner Join wird manchmal auch als Equi Join (Gleichheitsverknüpfung) bezeichnet. Die Inhalte der verknüpften Felder beider Tabellen sind gleich.

Schreibweise für den Join

From Käufer k INNER JOIN Verkauf v	ausführliche Schreibweise
From Käufer k JOIN Verkauf v	wenn Join nicht genauer spezifiziert: -> Inner Join
From Käufer k, Verkauf v	wenn nichts spezifiziert: -> Inner Join

Schreibweise für den Alias

Select Name, Id, ProduktID	
From Käufer, Verkauf	ohne Alias, nur möglich wenn keine Konflikte
Where K_ID = K_ID	

Select Käufer.Name, Käufer.Id, Verkauf.ProduktID	ausführliche Schreibweise
From Käufer, Verkauf	
Where Käufer.K_ID = Verkauf.K_ID	

Select k.Name, k.Id, v.ProduktID	mit Alias
From Käufer as k, Verkauf as v	
Where k.K_ID = v.K_ID	

Höhere Fachschule Uster Bildungszentrum Uster

Left Outer Join

```

Select  k.Name, k.K_ID, v.ProduktID
From    Käufer k LEFT OUTER JOIN Verkauf v
ON      k.K_ID = v.K_ID

```

Name	K_ID
Baar	1
Chai	2
Corets	3
Melia	4

K_ID	ProduktID
1	A2
1	B3
4	D1
3	R5

Result Set

Name	K_ID	ProduktID
Baar	1	A2
Baar	1	B3
Melia	4	D1
Corets	3	R5
Chai	2	NULL

Prinzip

- Schliesst alle Zeilen aus der linken Tabelle mit in die Ergebnistabelle mit ein, auch ohne Entsprechung (Übereinstimmung) in der rechten Tabelle. Fehlende Daten werden mit NULL aufgefüllt.
- Mit einer linken äusseren Verknüpfung arbeiten Sie, wenn Sie eine vollständige Liste aller Daten aus der linken Tabelle benötigen.
- In diesem Beispiel erhalten Sie eine Liste aller Käufer, auch von denen, die noch nichts aus dieser Verkaufstabelle gekauft haben. Eine innere Verknüpfung würde nur diejenigen Käufer zurück geben, die auch etwas gekauft haben.
- In manchen DBS wird das WHERE durch ON ersetzt.

Ein **RIGHT OUTER JOIN** arbeitet analog.

Ein **FULL OUTER JOIN** hat im Result Set alle Datensätze aus beiden Tabellen. Wo es zwischen den beiden Datensätzen keine Entsprechung gibt, werden die Daten mit NULL aufgefüllt.

Die Klauseln Left Outer Join bzw. Right Outer Join können mit Left Join bzw. Right Join abgekürzt werden.

Höhere Fachschule Uster Bildungszentrum Uster

Cross Join

```

Select  k.Name, k.K_ID, v.ProduktID
From    Käufer k, Verkauf v

```

Name	K_ID
Baar	1
Chai	2
Corets	3

K_ID	ProduktID
1	A2
2	D1

Result Set

Name	K_ID	ProduktID
Baar	1	A2
Baar	1	NULL
Chai	2	NULL
Chai	2	D1
Corets	3	NULL
Corets	3	NULL

Prinzip

- Ein Select über 2 oder mehr Tabellen ohne WHERE und ohne ON ergibt einen Cross Join.
- Als Ergebnistabelle erhält man das Kartesische Produkt (Kreuzprodukt) der beiden Tabellen.

Anwendungen

- Kartesische Produkte sind in der Regel nutzlos.
- Eine mögliche Anwendung ist das Generieren von Checklisten über alle möglichen Kombinationen von Zuständen -> Kombinatorik.
- Die Anzahl Datensätze in der Ergebnistabelle ergibt sich aus der Multiplikation der Anzahl Datensätze aus beiden Tabellen.

3.2.b.3 Auto Join

Höhere Fachschule Uster Bildungszentrum Uster

Selbst - Verknüpfung

Select a.Name as Arbeiter, b.Name as Boss
From Mitarbeiter a, Mitarbeiter b
Where a.ReportsTo = b.ID

Mitarbeiter a

ID	Name	ReportsTo
1	Bigler	2
2	Mosciatti	3
3	Steiner	NULL

Mitarbeiter b

ID	Name	ReportsTo
1	Bigler	2
2	Mosciatti	3
3	Steiner	NULL

Arbeiter Boss

Bigler	Mosciatti
Mosciatti	Steiner

Prinzip

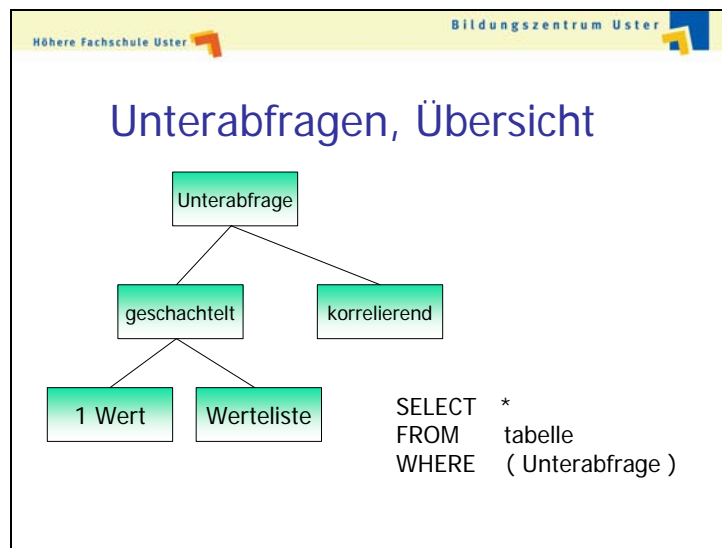
Wenn Sie Zeilen finden möchten, die mit anderen Zeilen derselben Tabelle übereinstimmende Werte aufweisen, müssen Sie die Tabelle mit einer anderen Instanz derselben Tabelle verknüpfen.

Beispiele

Jeder Angestellter ist einem Manager unterstellt, der seinerseits auch ein Angestellter ist. In einer Mitarbeitertabelle ist die MitarbeiterID der Primärschlüssel, während ReportsTo der Fremdschlüssel ist, über den die Tabelle mit sich selbst in Beziehung gesetzt wird.

Selbstverständlich gibt es viele weitere Anwendungen.

3.2.b.4 Subqueries



Prinzip

- Eine Unterabfrage (engl. Subquery) ist eine Abfrage, deren Ergebnis als Argument an die Hauptabfrage übergeben wird.
- Eine Unterabfrage kann anstelle eines Ausdrucks verwendet werden, solange ein einzelner Wert oder eine Werteliste zurückgegeben wird.

Geschachtelte Unterabfrage

- Ist eine Unterabfrage auch für sich alleine ausführbar, so ist sie unabhängig von der Hauptabfrage, man spricht in diesem Fall von einer geschachtelten Unterabfrage.
- Geschachtelte Unterabfragen geben entweder einen einzelnen Wert oder eine Werteliste zurück.

Korrelierende Unterabfrage

- Wenn eine Unterabfrage abhängig ist von der Hauptabfrage ist sie für sich alleine nicht ausführbar. Man spricht dann von einer korrelierenden Unterabfrage.

Unterabfrage versus Join

- Ein Select mit einer Unterabfrage kann sehr oft auch mit einer Verknüpfung von 2 Tabellen ausgeführt werden.
- Verknüpfungen sind oft schneller als Unterabfragen.
- Ob man eine Unterabfrage oder eine Verknüpfung realisiert hängt von den persönlichen Präferenzen ab.

Höhere Fachschule Uster Bildungszentrum Uster

Unterabfragen, geschachtelt

Einzelner Wert

```
SELECT orderID, name
FROM order
WHERE date = ( Select max(date) from order )
```

'2006-04-12 12:35'

Anwendung

- Unterabfragen, die einen einzelnen Wert zurückgeben, können überall dort verwendet werden, wo Ausdrücke gebraucht werden.
- Verwenden Sie die geschachtelte Unterabfrage in einer Where-Klausel mit einem Vergleichsoperator.
- Jede geschachtelte Unterabfrage wird nur einmal ausgewertet.
- Sie müssen Unterabfragen in Klammern setzen.
- Um eine Unterabfrage besser verstehen zu können, überlegt man sich, was sie zurückgibt und setzt dieses Resultset anstelle der Unterabfrage ein.

Beispiel

Dieses Beispiel gibt alle Kunden zurück, die Bestellungen am zuletzt aufgezeichneten Tag aufgegeben haben.

Höhere Fachschule Uster Bildungszentrum Uster

Unterabfragen, geschachtelt

Werteliste

```
SELECT name
FROM kunde
WHERE KundenID IN ( Select KundenID
                    From Order
                    Where date > '2006-05-12' )
```

(7, 23, 45, 69)

Anwendung

- Überprüfen Sie die Zugehörigkeit zu einer erzeugten Werteliste mit einer geschachtelten Unterabfrage, die eine Werteliste zurückgibt.
- Verwenden Sie in einer Where-Klausel den IN-Operator (oder ein anderer Mengen-Operator, wie z.B. Exists) in Verbindung mit Unterabfragen, die eine Werteliste zurückgeben.
- Jede geschachtelte Unterabfrage wird nur einmal ausgewertet.

Beispiel

Hier wird eine Liste aller Kunden erzeugt, die nach dem 5.12.99 Käufe getätigt haben.

Höhere Fachschule Uster Bildungszentrum Uster

Unterabfragen, korrelierend

```
SELECT      o.OrderID, o.CustomerID
FROM        orders o
WHERE       20 < ( SELECT      d.Quantity
                   FROM        details d
                   WHERE        o.OrderID = d.OrderID
                   AND          d.ProductID = 23 )
```

Prinzip

- Bei korrelierenden Unterabfragen verwendet die innere Abfrage Daten aus der Hauptabfrage.
- Die Unterabfrage wird einmal für jede in der Hauptabfrage zurückgegebene Zeile ausgeführt.

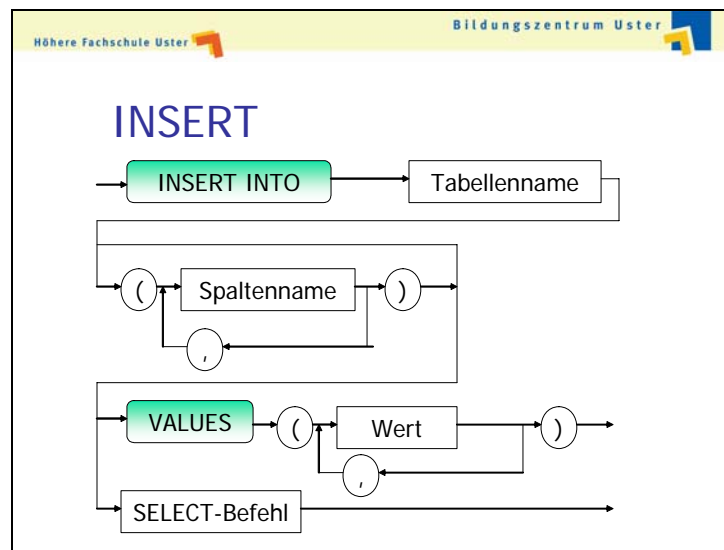
Korrelierende Unterabfrage versus Join

- Korrelierende Unterabfragen sind langsam.
- Korrelierende Unterabfragen können in der Regel zu Verknüpfungen umformiert werden.
- Die vorzugsweise Verwendung von Verknüpfungen anstelle korrelierender Unterabfragen ermöglicht dem SQL-Optimierer, selber die effizienteste Vorgehensweise zu bestimmen.

Beispiel

In diesem Beispiel wird eine Liste von Kunden zurückgegeben, die mehr als 20 Stück des Produkts mit Nummer 23 mit einer einzigen Bestellung geordert haben.

3.2.c Insert



Prinzip

Mit INSERT... VALUES ... wird eine neue Zeile mit konstanten Werten in eine Tabelle eingefügt.

Rahmenbedingungen

Werden keine Spaltennamen angegeben, müssen die Werte nach VALUES in Reihenfolge und Anzahl den Spalten der Tabelle entsprechen.

Ein Wert muss immer den gleichen Datentyp aufweisen, wie die entsprechende Spalte.

Nicht genannte Spalten werden mit NULL gefüllt.

Dynamisches Einfügen

Mit INSERT ... SELECT ... können Daten von einer Tabelle in eine andere eingefügt werden.

```
INSERT INTO  Tabellennamen
             ( spalte1, spalte2, ... )
SELECT      ...
FROM        ...
WHERE       ...
```

Temporäre Tabellen (SQL Server)

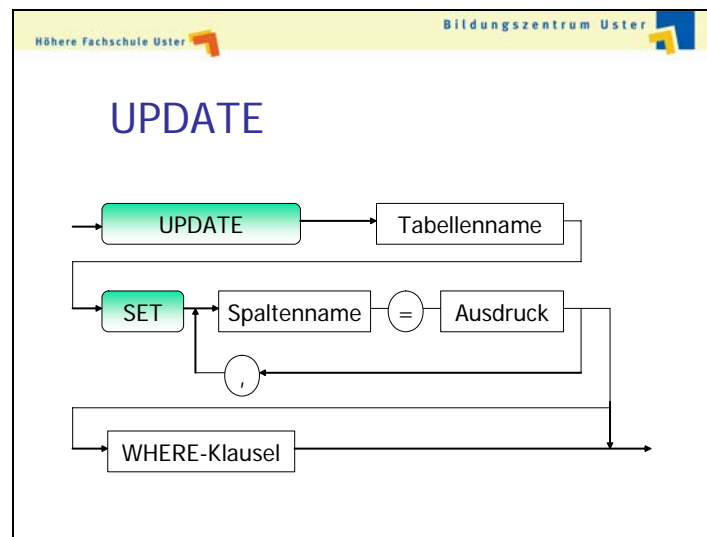
Mit SELECT ... INTO ... können Daten von einer bestehenden Tabelle in eine neue temporäre Tabelle eingefügt werden. Diese Option ist recht schnell, da das Transaction Log nicht geschrieben wird.

Temporäre Tabellen beginnen immer mit #. Temporäre Tabellen können nur von demjenigen, der sie erzeugt hat, auch angesprochen werden. Temporäre Tabellen sind ein ideales Speichermedium um Zwischenresultate zu speichern. Temporäre Tabellen werden nach Beendigung der Session automatisch gelöscht. (Andere DBS haben eine andere Syntax, Access führt diese Option nicht).

```
SELECT      m.*
INTO        #Bonus
FROM        Mitarbeiter m
WHERE       m.AnzahlVerkäufe > 100
```

```
Select      name
from        #Bonus
Where       ...
```

3.2.d Update



Prinzip

- Für jede Zeile der Tabelle, welche die Where-Klausel erfüllt, werden die Spalten durch die Werte der entsprechenden Ausdrücke ersetzt.
- Wird die Where-Klausel weggelassen, werden alle Zeilen erfasst.

Beispiel

Der Preis aller Sträucher wird um 5% erhöht. Die entsprechenden Felder der Tabelle Pflanze müssen entsprechend geändert werden.

```

UPDATE    pflanze
SET       preis = preis * 1.05
WHERE     sorte = 'STRAUCH'
  
```

Update mit Bedingungen über mehrere Tabellen

Die meisten DBS'e haben auch eine FROM – Klausel, damit Bedingungen über mehrere Tabellen gesetzt werden können.

3.2.e Delete

Syntax: DELETE From Tabellenname WHERE-Klausel

- Jede Zeile der Tabelle, welche die Where-Klausel erfüllt, wird gelöscht.
- Wird die Where-Klausel weggelassen, werden alle Zeilen aus der Tabelle gelöscht.
- Die Meta-Daten werden nicht gelöscht, dazu dient der Befehl DROP Table.

Wissenswertes

- Besteht eine Referenz eines andern Datensatzes auf einen zu löschenden Datensatz, so wird dieser nicht gelöscht, sofern das DBS die Referentielle Integrität überwacht.
- In kommerziellen Anwendungen wird recht wenig gelöscht, weil damit die Historisierung verloren geht. Stattdessen wird nur eine Statusänderung vorgenommen.

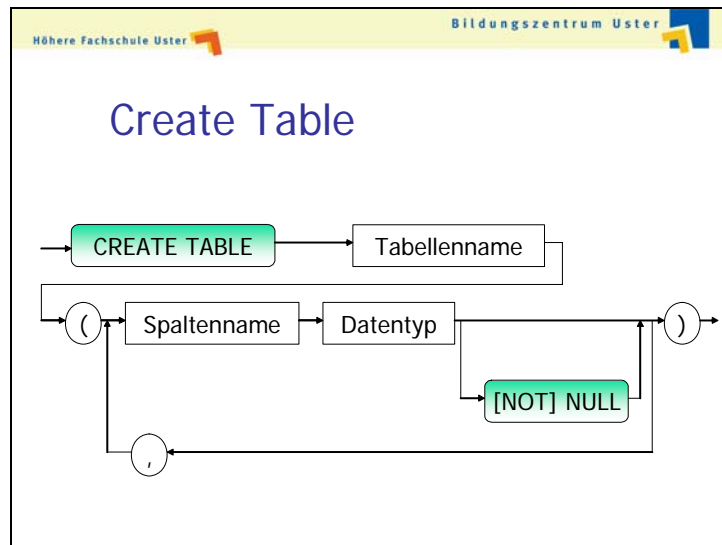
Beispiel: Alle Daten der Bestellung 0190 sind zu löschen.

```

DELETE
FROM    bestelldaten
WHERE   bestellnr = '0190'
  
```

3.3 DDL

3.3.a Create Table



Syntax

CREATE Table Mitarbeiter

```

(
    ID      int      not null,
    Name    Varchar(20) null,
    Eintritt Date     null
)
  
```

Datentypen

Bezeichnung und Syntax von Datentypen variieren von DBS zu DBS recht stark. Aber alle haben folgende skalare Typen auf mindestens eine Art implementiert:

- Char (n) fixe Anzahl Characters, fehlende Characters werden mit Spaces gefüllt
- Varchar(n) variable Anzahl Characters, bis maximal n Characters
- Int Integer
- Float, Real Gleitkommazahl
- Date Datum

MS SQL Server 2000

- 2 Milliarden Tabellen pro DB
- 1024 Spalten pro Tabelle
- 8060 Byte pro Datensatz

Eine bestehende Tabelle ändern

Mit ALTER TABLE Tabellenname ... können

- Spalten einer Tabelle entfernt oder
- neue Spalten zu einer Tabelle hinzugefügt

Tabelle löschen

- Um eine Tabelle wieder zu löschen, verwendet man DROP TABLE Tabellenname.
- Mit DROP TABLE werden nicht nur die Daten, sondern auch die Tabellendefinitionen und die zugehörigen Berechtigungen gelöscht.
- Sind Abhängigkeiten auf die zu löschenden Tabellen vorhanden, kann die Tabelle nicht gelöscht werden.

3.3.a.1 Large Objects

Höhere Fachschule Uster
Bildungszentrum Uster

Large Objects

```

Create Table Projekt_Vorschlag
(
    id            int            not null,
    Name          varchar(20)    not null,
    Beschreibung  CLOB           null,
    Budget_Kalk   BLOB           null,
    Deckblatt     BFILE          null
)
        
```




Idee

Es geht um die Ablage grosser Objekte, wie umfangreiche Texte, Bilder, Videos, Animationen, etc. Diese Objekte werden nicht als komplexe Strukturen betrachtet, sondern als einfache, aber grosse Datentypen.

BLOB

BLOB steht für binary large object. Das sind z.B. Grafiken, Videos aber auch ausführbarer Programmcode. Ein BLOB kann je nach System bis zu 4 GB Daten umfassen.

CLOB

Wenn es sich um grosse Mengen lesbarer Zeichen handelt, kann man sie in CLOB's ablegen: character large object. Auch hier sind Grössenordnungen bis zu 4 GB möglich.

BFILE

BLOB's und CLOB's werden im DBS selbst untergebracht. Die dritte Kategorie, die BFILE's sind als Dateien ausserhalb des DBS abgelegt.

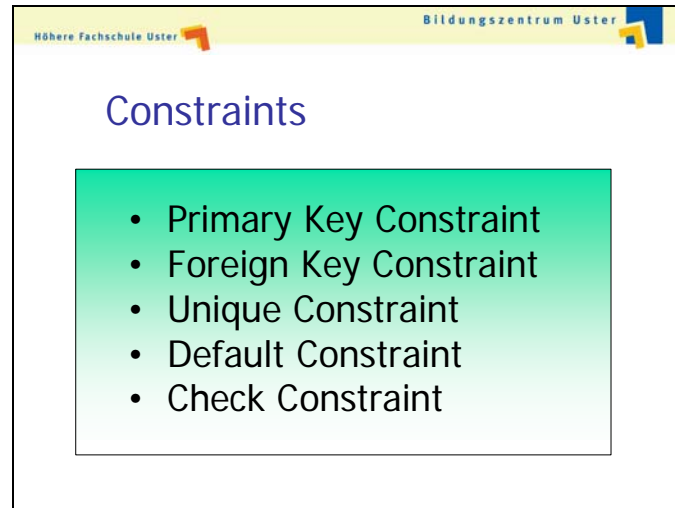
Beispiel

Obiges Beispiel definiert eine Tabelle, in der Projektvorschläge abgelegt werden. Die Projektbeschreibung ist ein umfangreicher Text. Er liegt in einem character large object. Die Projektkalkulation wurde z.B. mit einem Tabellenkalkulationsprogramm erstellt. Das alles wird binär in einem binary large object abgelegt. Und dann gibt es schliesslich in unserem Unternehmen ein Standardformular, das als Deckblatt für Projektvorschläge genutzt wird. Es liegt als Word-Datei irgendwo im Rechnersystem ausserhalb des Datenbanksystems.

SQL3

SQL3 definiert diese Datentypen erstmals als Standard. Die meisten grossen Datenbankhersteller hatten solche Datentypen aber schon lange vorher implementiert, zum Teil unter anderem Namen mit einer anderen Syntax.

3.3.a.2 Constraints



Prinzip

Nicht nur Programmierer machen Fehler, sondern vor allem auch Benutzer. Mit Constraints kann man sie vor sich selber schützen. Constraints sind Einschränkungen, die vom Programmierer definiert werden und deren Einhaltung von der DB erzwungen wird.

Primary Key

- Erzwingt Entitäts-Integrität (Einmaligkeit des Datensatzes)
- Spalte muss als NOT NULL definiert sein
- Kreiert automatisch einen Index
- Wird von REFERENCES Constraint als Referenzpunkt angesprochen
- Hat die gleiche Charakteristik wie Unique Constraint, ausser, dass er NULL nicht zulässt und pro Tabelle nur einmal vorkommen darf.
- Ex: Create Table lieferant (..., CONSTRAINT PK_Id PRIMARY KEY (id))

Foreign Key

- Foreign Keys Constraints garantieren, dass nur Werte eingefügt werden können, die sich bereits in der anderen Tabelle befinden.
- Umgekehrt verhindern sie die Löschung von Datensätzen in der referenzierten Tabelle, wenn noch entsprechende Bezüge in der referenzierten Tabelle existieren.
- Ex: Create Table Artikel (..., CONSTRAINT FK_Lieferant FOREIGN KEY (LieferantId) REFERENCES lieferant (id))

Unique

- Unique Constraints spezifizieren, dass zwei Zeilen nicht den gleichen Wert in der gleichen Spalte haben können.
- Erlaubt NULL
- Mehrere Unique Constraints können in einer Tabelle vorkommen.
- Ex: Create Table Angebot (..., CONSTRAINT U_code UNIQUE (lfr_code, art_code))

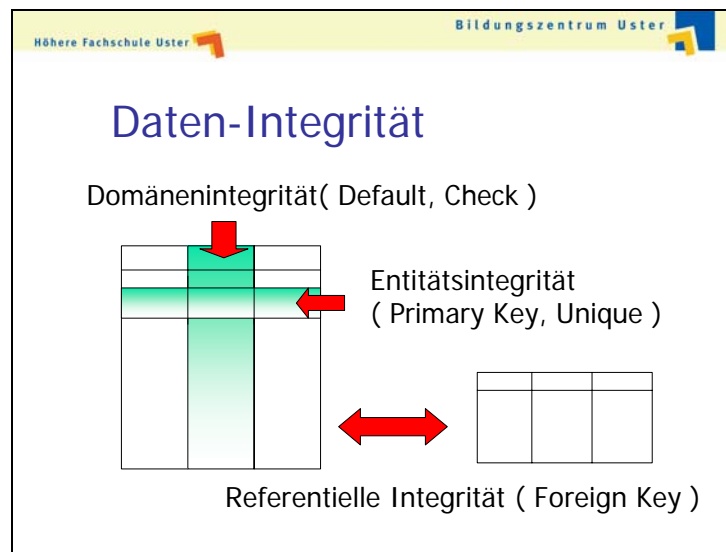
Default

- Der Default Constraint füllt einen Wert in ein Feld ein, wenn das Feld im INSERT-Befehl ausgelassen wird.
- Ex: Create Table Adult(..., CONSTRAINT D_state, DEFAULT 'CA' FOR state
- Zusätzlich zu Konstanten kann DEFAULT auch DBS-spezifische Funktionen beinhalten: current_user(), current_timestamp().

Check

- Ein Check Constraint bestimmt den Wertebereich, der eingegeben werden darf.
- Verstärkt die Integrität des Datentyps durch Limitierung der möglichen Werte.
- Wird jedes mal kontrolliert, wenn ein INSERT oder UPDATE gemacht wird.
- Ex: Create Table Adult(..., CONSTRAINT C_alter CHECK (alter between 1 and 120)

3.3.a.3 Daten-Integrität



Die verschiedenen Constraints lassen sich gruppieren in 3 Typen:

- Domänen-Integrität
- Entitäts-Integrität
- Referentielle-Integrität

Referentielle Integrität

Die referentielle Integrität ist in einer DB erfüllt, wenn jeder Foreign Key ungleich NULL eine Entsprechung beim zugehörigen Primary Key hat.

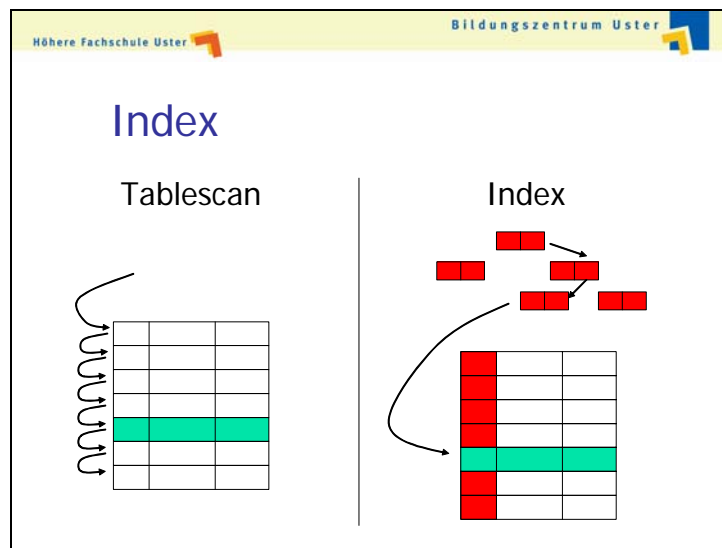
Ein DBS kann auf eine Integritätsverletzung auf drei Arten reagieren:

- **RESTRICT** Bei einer restriktiven Lösung wird die Löschaktion abgewiesen, die eine Verletzung der ref. Integrität zur Folge hätte.
- **CASCADE** Beim kaskadierenden Löschen werden alle Datensätze gelöscht, die den Schlüssel des gelöschten Datensatzes als Fremdschlüssel enthalten.
- **SET NULL** Als dritte Variante besteht die Möglichkeit, den Inhalt des Fremdschlüsselattributs auf NULL zu setzen.

Zur Durchsetzung der referentiellen Integrität bestehen Alternativen:

- als Foreign Key Constraint
- als Trigger (siehe Kapitel SQL)

3.3.a.4 Index



Datenzugriff

In einer DB werden 2 Methoden zum Zugriff auf Daten eingesetzt: Tablescan oder indizierter Zugriff. Bei einer Abfrage prüft die DB zunächst, ob ein Index vorhanden ist. Anschliessend wird mit dem Abfrageoptimierer (die Komponente für die Generierung optimaler Ausführungspläne) ermittelt, ob durch einen Tabellenscan oder durch die Verwendung des Index ein effizienterer Datenzugriff ermöglicht wird.

Tablescan

- Der Startpunkt ist der erste Datensatz in der Tabelle.
- Scans erfolgen von Datensatz zu Datensatz.
- Jeder Datensatz in der Tabelle wird gelesen, und die den Abfragekriterien entsprechenden Zeilen werden extrahiert.
- Tablescans sind empfehlenswert für den Zugriff auf kleine Tabellen.

Index

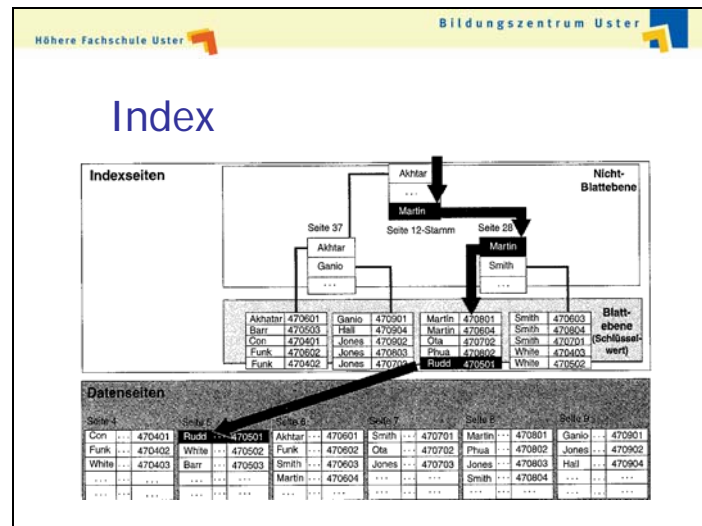
- Ziel und Zweck eines Indexes ist die Erhöhung der Performance einer Abfrage.
- Eine DB setzt Indices ähnlich ein, wie ein Leser den Index (oder Inhaltsverzeichnis) eines Buches verwendet. Um in einem Buch Infos zu einem bestimmten Thema zu finden, können Sie im Index am Ende des Buches nach dem Thema suchen. Im Index werden die Stichworte des Buches zusammen mit Verweisen auf die jeweilige Seite aufgelistet. So sparen Sie ziemlich viel Sucharbeit.
- Ein Index ist eine interne Tabelle von Zeigern auf bestimmte Zeilen mit bestimmten Spalten.

Kreieren eines Index

- Ein Index wird auf einer Kolonne einer Tabelle kreiert.
- `CREATE INDEX indexname ON tabellenname (kolonnenname)`

Grundsätze

- Erstellen Sie Indices für häufig durchsuchte Spalten wie Primärschlüssel, Fremdschlüssel oder andere Spalten, die zum Verknüpfen von Tabellen eingesetzt werden.
- Indices beanspruchen Speicherplatz und führen zu erhöhten Verwaltungs- und Wartungskosten.
- In der Regel sind Indices für kleine Tabellen weniger empfehlenswert, da das Durchlaufen der Indexseiten aufwendiger sein kann als das Scannen der gesamten Tabelle.
- Spalten, auf die in einer Abfrage nur selten verwiesen wird, sollten nicht indiziert werden. Ebenso wenig wie Spalten, die nur wenig eindeutige Werte enthalten (ex: männlich oder weiblich). Hier bietet die Indizierung keine Vorteile.

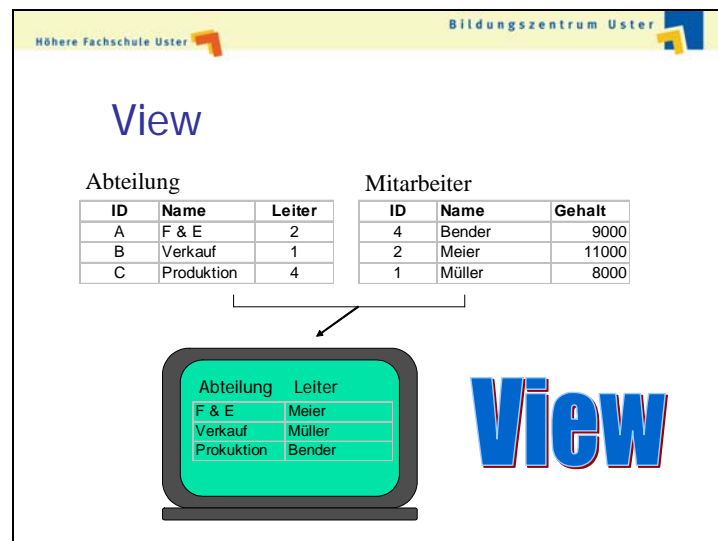


In obiger Abbildung wird dargestellt, wie für das folgende Query die Suche im Indexbaum erfolgt. Selbstverständlich gibt es viele andere Varianten.

```
SELECT    lastname, firstname
FROM      member
WHERE     lastname = 'Rudd'
```

1. Das DBS stellt fest, dass für die Spalte lastname ein Index vorhanden ist, der zum Abrufen von Zeilen geeignet ist, die den Nachnamen Rudd enthalten.
2. Die Suche auf Nicht-Blattebene beginnt im Wurzelknoten. Der letzte Schlüsselwert auf dieser Seite (Martin) ist kleiner als der Suchwert Rudd. Die Suche wird auf der Seite fortgesetzt, auf die dieser Schlüsselwert verweist.
3. Die Suche wird auf der Nicht-Blattebene der Indexseite (Seite 28) fortgesetzt. Der Suchwert Rudd liegt auf dieser Seite zwischen den Schlüsselwerten Martin und Smith. Die Suche wird auf der Seite fortgesetzt, auf die der erste dieser Schlüsselwerte verweist.
4. In diesem Schritt hat die Suche die Blattebene erreicht. Hier wird die Index-Seite nach der Index-Zeile durchsucht, deren Schlüsselwert mit dem Suchbegriff Rudd übereinstimmt. Die Zeilen-ID 470501 ist ein logischer Verweis auf den gesuchten physikalischen Datensatz.

3.3.b View



Idee

Viele Tabellen müssen für die Darstellung am Bildschirm und für Auswertungen wieder zusammengeführt werden. Dieser Vorgang kann auf 2 Arten realisiert werden:

- Programmierung in **jedem** Programmteil, welcher die entsprechende Darstellung der Daten benötigt.
- Erstellen **einer** View, welche von **allen** Programmteilen verwendet werden kann.

Programmierung

CREATE VIEW Abteilungsübersicht AS

```
Select      a.Name as Abteilung, m.Name as Leiter
From        Abteilung a, Mitarbeiter m
Where       a.Leiter = m.ID
```

```
Select      *
From        Abteilungsübersicht
```

Betrachtungsweise

- Eine View ist ein gespeichertes Select-Statement.
- Eine View kann als virtuelle Tabelle betrachtet werden.

Vorteile

- Eine View bietet alle Vorteile einer einfachen Subroutine, einmal programmieren, vielfach anwenden.
- Eine View reduziert für deren Anwender die Komplexität, die sich darunter verbirgt.
- Eine View kann als Komponente des Datenschutzes eingesetzt werden, indem die Existenz von gewissen Zeilen oder Spalten (hier das Gehalt) je nach Benutzergruppe ein- oder ausgeblendet werden kann. Ein DBS kann den Benutzern keine Berechtigung zum Abfragen bestimmter Spalten in Basistabellen gewähren, sondern nur eine Berechtigung für Views.

3.4 DCL

3.4.a Transaction

Höhere Fachschule Uster Bildungszentrum Uster

Transaction: Übersicht

Eine Transaktion ist eine Folge von Operationen, die

A	Atomar
C	Konsistent
I	Isoliert
D	Dauerhaft

sein muss.

Das bisher Gesagte reicht im Wesentlichen aus, wenn wir eine Datenbank auf einem Einzelplatzrechner betreiben. In betrieblichen Anwendungen ist jedoch davon auszugehen, dass die Daten unternehmensweit organisiert sind und die Nutzer Zugriff über ein lokales Netzwerk haben.

Ausgangslage

Sobald mehrere Benutzer **gleichzeitig** auf dieselben Datenbestände zugreifen, müssen spezielle Massnahmen getroffen werden, um Probleme wie Inkonsistenzen oder Deadlocks zu vermeiden. In Verarbeitungen werden meist mehrere Datensätze verändert; dabei lassen sich oft Sequenzen identifizieren, die entweder insgesamt korrekt abgeschlossen werden müssen oder sich überhaupt nicht im Datenbestand niederschlagen dürfen. Um dies sicherzustellen, werden derartig zusammengehörige Datenmanipulationen zu Transaktionen zusammengefasst.

Eine Transaktion hat 4 wesentliche Eigenschaften:

Atomarität

Atomarität besagt, dass eine Transaktion eine unteilbare Einheit darstellt. Sie wird entweder komplett durchgeführt oder gar nicht.

Konsistenz

Bei Transaktionsende müssen alle Konsistenzbedingungen erfüllt sein. Während der Transaktion können sie verletzt sein.

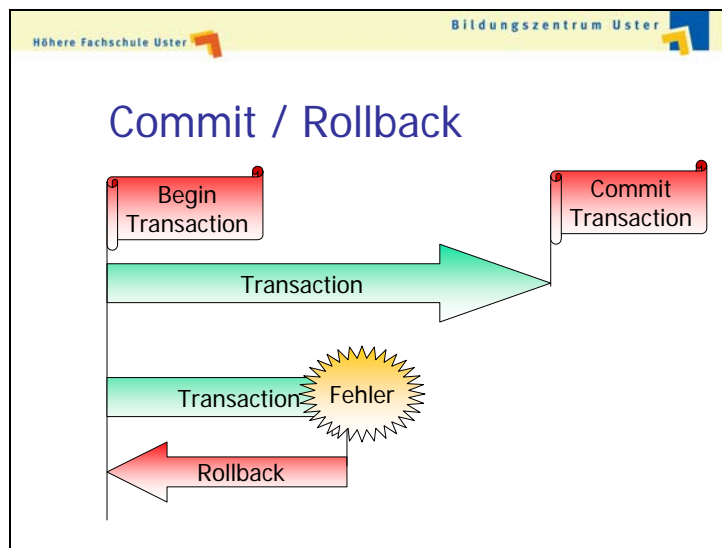
Isolation

Das Prinzip der Isolation verlangt, dass gleichzeitig ablaufende Transaktionen dieselben Resultate wie im Falle einer Einzelbenutzerumgebung erzeugen müssen; sie dürfen sich nicht gegenseitig beeinflussen. Die Transaktion bildet so eine Einheit für ihre Serialisierbarkeit.

Dauerhaftigkeit

Bei Programmfehlern, Systemabbrüchen oder Fehlern auf dem Speichermedium garantiert die Dauerhaftigkeit die Wirkung einer korrekt abgeschlossenen Transaktion. Alle mit commit abgeschlossenen Transaktionen müssen auch nach einem Systemabbruch auf der Harddisk 'verewigt' sein. Die Transaktion bildet eine Einheit für eine Wiederherstellung (Recovery).

3.4.a.1 Commit / Rollback



Eine Transaktion wird zu einem bestimmten Zeitpunkt mit **Begin Transaction** explizit begonnen und wird am Ende mit **Commit Transaction** abgeschlossen oder mit **Rollback Transaction** zurückgesetzt.

Etliche DBMS, die für den PC verfügbar sind, enthalten keine Mechanismen für die Transaktion. Dieses Fehlen der sehr komplizierten Mechanismen ist u.a. der Grund für den relativ tiefen Preis dieser Systeme. Solche Systeme sind daher nur für den Einbenutzerbetrieb zu verwenden. Bei der Auswahl eines DBMS ist auf diesen Punkt besonders Wert zu legen.

Transaktionsabbrüche

In grossen DBS, bei denen mehrere hundert Transaktionen pro Sekunde ausgeführt werden, sind Transaktionsabbrüche an der Tagesordnung. Solche Abbrüche sind entweder lokaler Natur, wenn nur eine Transaktion betroffen ist, oder sie sind global, wenn mehrere Transaktionen betroffen sind.

Die Ursachen für lokale Abbrüche sind z.B:

- Arithmetik-Fehler (Division durch 0)
- Canceln einer Transaktion durch den Benutzer
- Software-Fehler

Globale Abbrüche:

- Hardware-Fehler in CPU, BUS, etc.
- Harddisk-Crash
- Stromausfall
- Software-Fehler

Eine Transaktion wird durch ein Rollback zurückgesetzt (ungeschehen gemacht).

Rollback

Technisch liegt dieser Möglichkeit eine zeitweise Duplizierung der Daten zugrunde. Alle Tupel werden vor der Änderung in eine 'Befor-Image-Datei' kopiert. Aus dieser werden sie bei einem Rollback in die Datenbank zurück kopiert.

3.4.a.2 Lock-Mechanismen



Lock-Granularität

Jedes professionelle DBS bietet die Möglichkeit, Objekte einer Datenbank für Transaktionen zu locken (sperrern). Solche Locks können unterschiedliche Feinheiten besitzen, siehe oben.

Lockmanager

Die Anforderung, ein Objekt zu locken, wird an den so genannten Lockmanager gerichtet. Dieser Prozess führt die Lockanforderungen zentral für alle Transaktionen eines DBS aus. Er hält die Locks in speziellen Locktabellen, in denen neben der Transaktions-ID auch die eindeutige Kennung des gelockten Objekts eingetragen wird.

Bevor ein Objekt gelockt werden kann, muss der Lockmanager in der Locktabelle nachsehen, ob das Objekt bereits von einer anderen Transaktion gelockt wurde. Falls das der Fall ist, wird die den Lock anfordernde Transaktion in einen Wartezustand gesetzt, bis der Lock freigegeben ist.

In der Praxis werden zwei Arten von Locks unterschieden: exklusive und shared Locks.

Exklusive Locks

XLOCK's werden gesetzt, wenn eine Transaktion ein Objekt ändern möchte.

Shared Locks

Es muss sichergestellt sein, dass zu der Zeit, in der Transaktionen lesend auf Daten zugreifen, keine andere Transaktion diese Daten ändert; lediglich lesender Zugriff ist erlaubt.

Um dies zu gewährleisten, müssen auch lesende Transaktionen Daten locken. Aus diesem Grund gibt es einen andern, schwächeren Lock, den sogenannten Shared Lock (SLOCK). Damit wird sichergestellt, dass beliebig viele Transaktionen lesend auf ein Datenbankobjekt zugreifen können.

DeadLock

Ein Deadlock entsteht, wenn Transaktionen wechselseitig aufeinander warten oder wenn zyklische Abhängigkeiten vorliegen. DBMS verfügen meist über Algorithmen, die solche Verklemmungen aufspüren können. Aufgrund verschiedener Kriterien wird anschliessend entschieden, welche Transaktion zurückgesetzt wird, womit sich die Verklemmung auflöst. Ein sehr einfacher Entscheid ist z.B. aufgrund einer Zeitschranke möglich, d.h. nach Ablauf der Zeit wird eine der Transaktionen automatisch abgebrochen.

3.4.a.3 Isolation Levels

Isolation Levels				
Isolation Level	Effekte			
	Lost update	Dirty Read	Non Repeatable Read	Phantom
0: Read Uncommitted	nicht erlaubt	erlaubt	erlaubt	erlaubt
1: Read Committed	nicht erlaubt	nicht erlaubt	erlaubt	erlaubt
2: Repeatable Read	nicht erlaubt	nicht erlaubt	nicht erlaubt	erlaubt
3: Serializable	nicht erlaubt	nicht erlaubt	nicht erlaubt	nicht erlaubt

Zur Synchronisation des Mehrbenutzerbetriebs bietet SQL/92 über SET TRANSACTION ISOLATION LEVEL vier abgestufte Isolationsebenen an. Die unterschiedlichen „Isolation Levels“ lassen unterschiedliche Effekte bei der Transaktionsverarbeitung zu bzw. schliessen sie aus. Jeder Isolation Level schliesst bestimmte Phänomene, die im Mehrbenutzerbetrieb auftreten können, aus bzw. nimmt sie in Kauf. In konkurrierenden Transaktionen können folgende Phänomene auftreten:

Lost Update

Ein Lost Update wurde von einem anderen Update überschrieben, und geht somit verloren.

Dirty Read

Ein Dirty Read ist ein Lesevorgang, der veränderte Zeilen anderer noch nicht terminierter Transaktionen liest. Es können sogar Zeilen gelesen werden, die nicht existieren oder nie existiert haben. Die Transaktion sieht einen temporären Schnappschuss der Datenbank, der zwar aktuell ist, aber bereits inkonsistent sein kann. Offensichtlich erfasst die Abfrage Ergebnisse einer nicht (mit COMMIT) bestätigten Transaktion und erzeugt dadurch ein falsches Ergebnis.

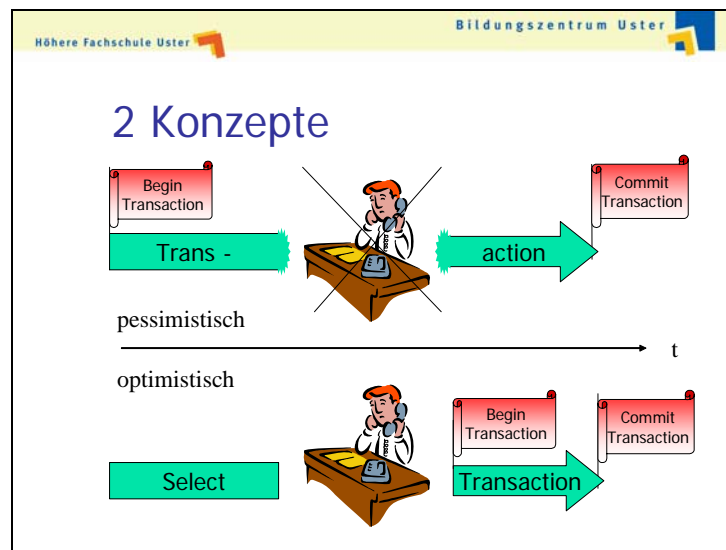
Non-Repeatable Read

Ein Non-Repeatable Read ist ein Lesevorgang, der im Falle von mehrmaligem Lesen zu unterschiedlichen Ergebnissen führt. Innerhalb einer Transaktion führt die mehrfache Ausführung einer Abfrage zu unterschiedlichen Ergebnissen, die durch zwischenzeitliche Änderungen (update) und Löschungen (delete) entstehen. Die einzelnen Abfrageergebnisse sind konsistent, beziehen sich aber auf unterschiedliche Zeitpunkte.

Phantom-Read

Ein Phantom ist ein Lesevorgang bzgl. einer Datenmenge, die einer bestimmten Bedingung genügen. Fügt eine andere Transaktion einen Datensatz ein, der ebenfalls diese Bedingung erfüllt, dann führt die Wiederholung der Abfrage innerhalb einer Transaktion zu unterschiedlichen Ergebnissen. Bei jeder Wiederholung einer Abfrage innerhalb einer Transaktion enthält das zweite Abfrageergebnis mehr Datensätze als das erste, wenn in der Zwischenzeit neue Datensätze eingefügt wurden.

3.4.a.4 2 Konzepte



Pessimistische Verfahren

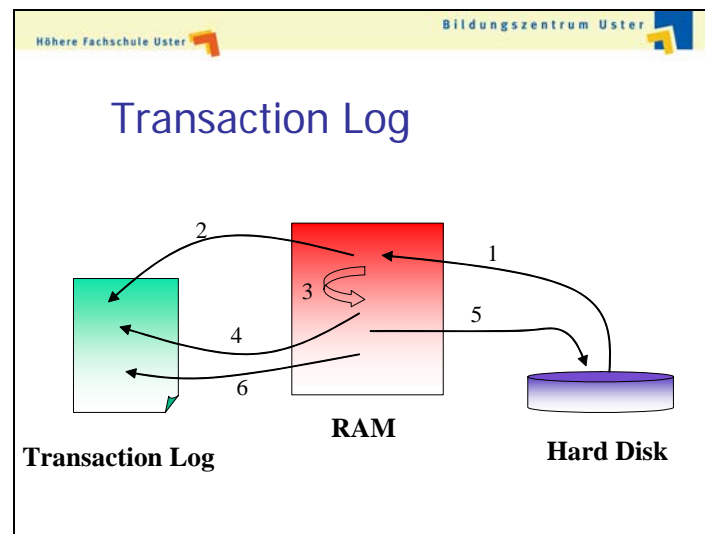
Pessimistische Verfahren sichern Transaktionen ab, indem sie durch Sperren die zu lesenden oder zu verändernden Daten vor andern Zugriffen schützen. Dabei werden zu Beginn der Transaktion alle Sperren gesetzt und diese am Ende der Transaktion wieder abgebaut. Transaktionen, die auf Daten zugreifen wollen, welche durch eine andere Transaktion gesperrt wurden, müssen warten, bis die Daten wieder freigegeben werden. Bei Transaktionen, die Benutzereingriffe erwarten, um beispielsweise Entscheidungen zu treffen, wird von diesem Verfahren abgeraten, da der Benutzer unbewusst andere Benutzer stundenlang blockieren kann.

Optimistische Verfahren

Bei optimistischen Verfahren geht man davon aus, dass Konflikte konkurrierender Transaktionen selten vorkommen. Daher wird wo immer möglich auf Sperren verzichtet, um so die Parallelität zu erhöhen und die Performance des Systems zu verbessern. Das Verfahren durchläuft 3 Phasen:

- **Lesephase:** In der Lesephase werden die benötigten Daten gelesen und in einen lokalen Arbeitsbereich kopiert und dort bearbeitet. Sind auch Benutzereingriffe notwendig, finden sie jetzt statt. Nichts ist gesperrt.
- **Validierungsphase:** In der Validierungsphase wird überprüft, ob andere Transaktionen in der Zwischenzeit dieselben Daten bearbeitet haben. Wenn ja, muss unter Umständen gar der Benutzer benachrichtigt werden. Wenn nein, kann in die Schreibphase übergegangen werden.
- **Schreibphase:** Jetzt wird die Transaktion ausgeführt. Die Daten sind nur für einen relativ kurzen Zeitintervall gesperrt.

3.4.a.5 Transaction Log



Motivation

Computersysteme stürzen ab. Wenn ein System abstürzt, während Datenbanktransaktionen aktiv sind, wird die Datenbank beschädigt. Man spricht von einer korrupten Datenbank, oder von korrupten Daten. Jedes DBS, das für ernsthafte Anwendungen eingesetzt werden soll, muss in der Lage sein, sich von möglichen Schäden zu erholen.

Idee

Der Schlüssel zur Wiederherstellung (Recovery) nach einem Absturz besteht darin, dass sämtliche Transaktionen in einem File protokolliert sind -> Transaction Log

Location

Das Transaction log muss zwingend auf einem physisch anderen Speichermedium sein als der Datenbestand.

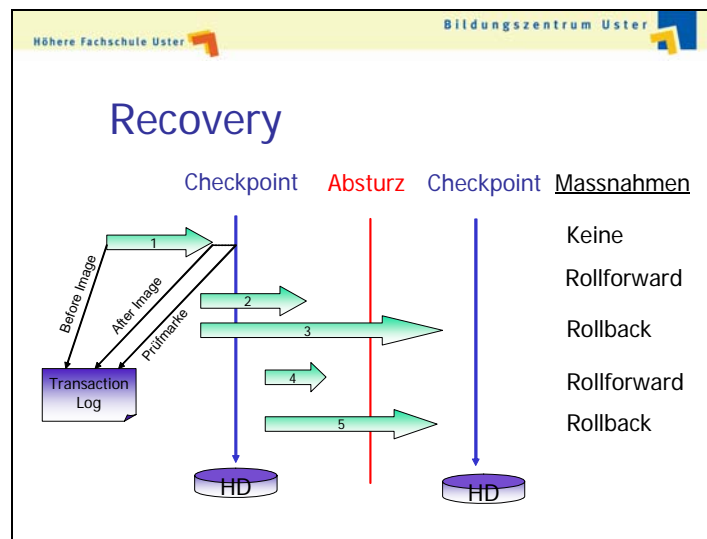
Checkpoint

Das Lesen und vor allem das Schreiben einer Harddisk ist langsam. Das heisst, diese HD-Zugriffe müssen optimiert sein. Darum wird nicht nach jeder Transaktion diese auch sofort auf den Datenträger geschrieben. Es wird nur periodisch, zu bestimmten Zeitpunkten – dies ist der Checkpoint – der gesamte Datencache en bloc auf die HD zurückgeschrieben.

Vorgehen

- 1) Datensatz wird von der HD ins RAM gelesen.
- 2) Datensatz wird als `Before Image` ins Transaction log geschrieben.
-> ermöglicht Rollback
- 3) Transaktion wird im RAM ausgeführt.
- 4) Veränderter Datensatz wird als `After Image` ins Transaction log geschrieben.
-> ermöglicht Rollforward
- 5) Beim nächsten Checkpoint wird veränderter Datensatz vom RAM auf die HD geschrieben.
- 6) Eine Prüfmarke wird für diese Transaktion ins Transaction Log geschrieben, der aussagt, dass die Transaktion sich auf der HD widerspiegelt.

3.4.a.6 Recovery



Generell

- Ein DBMS stellt sicher, dass sämtliche Transaktionen, für die ein Commit ausgeführt wurde, bei einem Stromausfall, bei SW-Fehlern und andern Abstürzen wiedergegeben werden.
- Nach einem Absturz wird beim Booten anhand des Transaction Logs für alle Transaktionen, für die ein Commit vorliegt, ein Rollforward und für alle unvollständigen Transaktionen ein Rollback ausgeführt.
- Um die verlorenen Transaktionen zu wiederholen, muss nur bis zum letzten Checkpoint zurückgegangen werden.

Beispiel

- Transaktion 1 hat im Transaction log einen Prüfpunkt, d.h. die Transaktion wird auch nach dem Absturz auf der DB angezeigt. -> keine Massnahmen
- Für die Transaktionen 2 und 4 wurde der Commit nach dem Checkpoint ausgeführt, d.h. sie haben keinen Prüfpunkt im Transaction log. Diese Transaktionen müssen beim Rebooten der DB anhand des Transaction Logs wieder hergestellt werden (Rollforward).
- Die Transaktionen 3 und 5 haben kein After Image in Transaction log, d.h. das DBS nimmt beim Rebooten einen Rollback vor.

Mirroring

Man spricht von einer Spiegelung einer DB, wenn zwei separate Kopien des Datenbestandes auf zwei verschiedenen nichtflüchtigen Speichermedien verwaltet werden. Jedes Mal, wenn eine Kopie geändert wird, wird gleichzeitig die andere Kopie geändert. Auf diese Weise verlieren Sie – falls eine Ihrer Festplatten crashed – nicht nur keine Daten, sondern auch keine Zeit, weil die Verarbeitung mit Hilfe der Spiegelplatte fortgesetzt werden kann.

Schlussfolgerung

Die Verfahren zur Wiederherstellung einer DB nach einem Absturz, wie mit Hilfe eines Transaction Logs sind wirksam, aber teuer. Die Spiegelung ist sogar noch teurer. Bei Datenbankanwendungen, von denen das Überleben eines Unternehmens abhängt, geht man davon aus, dass die Verbesserung der Zuverlässigkeit, die mit diesen Techniken erreicht werden, die Kosten wert sind. Bei preiswerten DBS wie z.B. Access werden Sie solche Funktionen jedoch nicht finden.

3.4.b Datenschutz

Höhere Fachschule Uster Bildungszentrum Uster

Datenschutz, Grundsätze

Positives Schutzsystem:

- Niemand kann Daten ansehen oder verändern, wenn er nicht ausdrücklich dazu befugt ist.
- Jedes DB-Objekt hat einen Eigentümer. Das ist der Benutzer, der es kreiert hat.
- Nur der Eigentümer selbst kann mit seinen Objekten arbeiten. Will er ändern das Manipulieren seiner Tabellen erlauben, muss er dies explizit mit GRANT tun.

Datenschutz

Datenschutz ist der Schutz von Daten vor unberechtigtem Zugriff und Gebrauch. Dieser Begriff ist klar vom Begriff Datensicherheit zu trennen; Datensicherheit ist die Sicherheit vor Datenverlust durch technische und organisatorische Massnahmen.

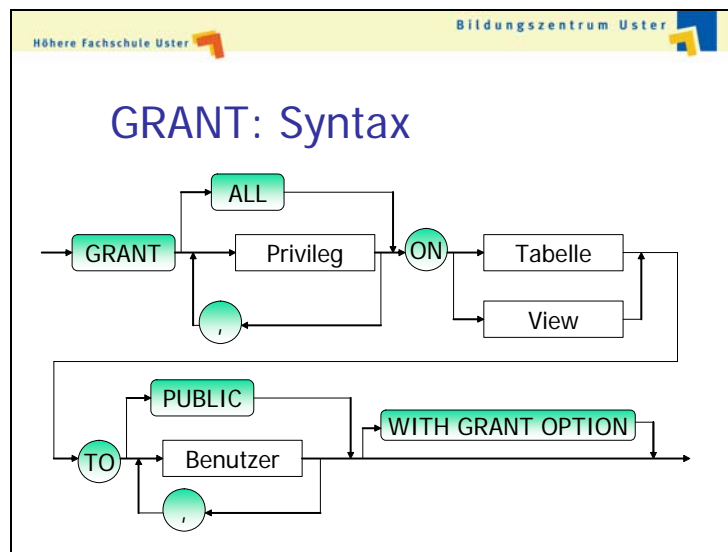
Privilegien

Privilegien sind auf 2 verschiedenen Ebenen angesiedelt:

- Der Zugang zur DB ist durch allgemeine Zugangsprivilegien geregelt. Diese sind herstellerabhängig realisiert und nicht Gegenstand des SQL-Standards. Der Standard geht davon aus, dass jeder Benutzer der DB mit einer UserID und einem Passwort ausgestattet ist und dass den DB Benutzern, die keine derartige Autorisierung vorweisen können, jeglichen Zugriff auf alle Funktionen verweigert wird.
- Die Rechte zur Manipulation einzelner DB-Objekte (Tabellen, Views, etc.) werden für jeden Benutzer mit objektbezogenen Privilegien verwaltet. Dazu dienen die Befehle GRANT und REVOKE.

DB-Systeme enthalten oft sensible Informationen, die nicht für jeden verfügbar sein sollten. SQL bietet verschiedene Zugriffsebenen – von keinem bis zu totalem Zugriff - mit mehrere Zwischenstufen.

Identifikation	Anmeldung eines Benutzers beim System unter Angabe einer Benutzerkennung
Authentisierung	Prüfvorgang durch das System, mit dem sichergestellt werden soll, dass ein Benutzer, der sich mit einer bestimmten Kennung identifiziert, auch tatsächlich dieser Berechtigte ist. Dies wird heute noch meist mit einem Passwort realisiert.
Autorisierung	Akt der Vergabe von Zugriffsrechten an einen Benutzer



Erklärung

PUBLIC ist die Gesamtheit der Benutzer, die Zugang zu einer bestimmten DB haben.

WITH GRANT OPTION bedeutet, dass der Benutzer die erhaltenen Privilegien wiederum mittels GRANT weitergeben kann.

Revoke

Mit Revoke können Berechtigungen wieder entzogen werden.

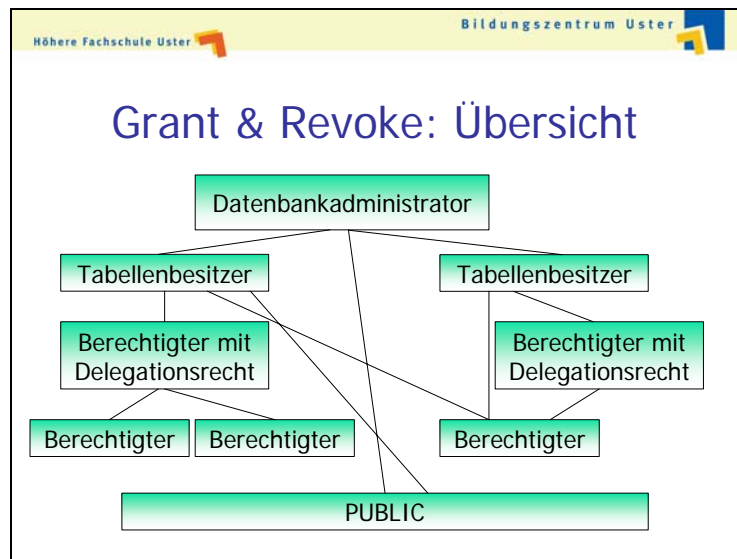
Beispiele

```
GRANT      SELECT, INSERT
ON         Bestelldaten
TO         PUBLIC
```

```
GRANT ALL
ON         Bestelldaten
TO         marianne
WITH GRANT OPTION
```

```
REVOKE     ALL
ON         Bestelldaten
FROM       marianne [RESTRICT] | [CASCADE]
```

Der optionale Zusatz RESTRICT bedeutet, dass die Revoke-Anweisung nicht durchgeführt wird, wenn der betreffende Nutzer seine Privilegien an andere weitergegeben hat. Durch die Angabe von CASCADE werden auch alle weitergegebenen Privilegien zurückgenommen.



Datenbankadministrator

- Die höchste Autorität in einer DB ist der Datenbankadministrator. Andere Namen sind: DBA, Systemadministrator, Superuser.
- Der DBA hat alle Rechte und Berechtigungen für alle Aspekte der DB.
- Die Position eines DBA ist mit Macht, aber auch mit Verantwortung verbunden. Mit dieser Macht können Sie Ihre Datenbank leicht beschädigen und Tausende von Arbeitsstunden zunichte machen. Ein DBA muss sich die Konsequenzen seiner Aktionen sorgfältig überlegen.
- Ein DBA kontrolliert auch die Rechte der andern Benutzer. Er kann beispielsweise besonders zuverlässigen Mitarbeitern grössere Zugriffsrechte einräumen als der Mehrzahl der andern Benutzer.

Der DBA hat folgende Aufgaben:

- Installation und Updates des DBS
- Konfiguration von Server und Clients
- Sichern und Wiederherstellen eines DBS
- Überwachung der Systemauslastung und Reaktion auf bestimmte Alarme
- Behebung von Systemproblemen
- Ansprechpartner für Programmierer und Benutzer
- Überwachung von HD Spaces und Installation neuer HD's

Tipps

- Der beste Weg, DBA zu werden, besteht darin, das DBMS selbst zu installieren. Dabei nennt Ihnen das Handbuch einen Benutzernamen und ein Passwort. Ihre erste Aktion nach Ihrer offiziellen Anmeldung sollte darin bestehen, Ihr Passwort zu ändern. Denn jeder der das Handbuch liest und sich mit dem Benutzernamen und dem Passwort des DBA anmeldet, gilt für das System als DBA.
- Selbst wenn Sie über DBA-Berechtigungen verfügen, sollten Sie sich nur als DBA einloggen, wenn Sie spezielle Aufgaben ausführen wollen, für die DBA-Berechtigungen erforderlich sind. Wenn Sie damit fertig sind, loggen Sie sich aus, und loggen Sie sich dann für Ihre normalen Arbeiten mit Ihrem normalen Namen und Passwort wieder ein. Dies ist nur ein Selbstschutz.

Rollen und Gruppen

Die Verwaltung von Benutzern kann in grösseren Organisationen recht aufwendig sein. Einzelne DBS'e haben daher Konzepte wie Rollen (Zusammenfassungen von Privilegien) und / oder Gruppen (Zusammenfassungen von Benutzern).



Mit den Werkzeugen der DCL bietet das DBS unterschiedliche Schutzmechanismen an:

- Schutz vor unautorisiertem Zugriff
- Schutz vor überlappenden Aktionen bei Multi-User-Betrieb
- Schutz von Strom- und Geräteausfall

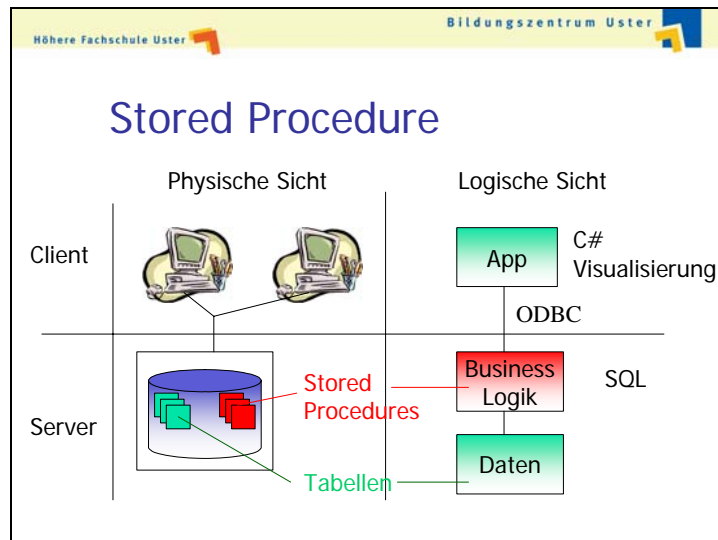
Richtig eingesetzt sind die Sicherheitswerkzeuge von SQL mächtige Schutzfaktoren wichtiger Daten. Falsch eingesetzt können dieselben Werkzeuge zu frustrierenden Störfaktoren werden, die legitime Benutzer bei ihrer Arbeit behindern.

Datenschutzgesetz

Zweck	Dieses Gesetz bezweckt den Schutz der Persönlichkeit und der Grundrechte von Personen, über die Daten bearbeitet werden.
Anwendbar	auf alle sensiven Daten mit direktem oder indirektem Personenbezug
Sensitive Daten	besonders schützenswerte Daten: Gesundheit, Intimsphäre, strafrechtliche Verfahren, Religion, politische Tätigkeiten, Lohn gehört nicht dazu
Datenschutzbeauftragter	wird vom Bundesrat gewählt überwacht Einhaltung des DSG berät private Personen und Firmen in Fragen des Datenschutzes
Umsetzung im Betrieb	techn. und org. Massnahmen treffen, damit Unberechtigte keinen Zugriff auf sensitive Daten haben Verpflichtung der MA in der Personalabteilung auf das Datengeheimnis (Bewerbungsunterlagen, Personaldossiers)

3.5 Procedurale Elemente

3.5.a Stored Procedure



Bisher haben wir interaktiv mit SQL gearbeitet. Auf diese Weise kann man zwar lernen, wie SQL funktioniert, aber nicht, wie es praktisch eingesetzt wird. Die typischen SQL-Anwender sind Applikationsentwickler, die ihren Lebensunterhalt nicht damit verdienen, interaktiv Datenbank-Abfragen an einem Terminal einzugeben. Hier kommen wir zum Begriff der Stored Procedure. Bei einer Stored Proc handelt es sich um eine benannte Zusammenstellung von SQL-Statements, die auf der DB gespeichert ist.

Das Konzept der Stored Procedures muss der professionelle DB-Programmierer auf jeden Fall beherrschen. Eine Stored Proc enthält viele Prozedurale Elemente. Wir entfernen uns hier von der Kernaussage, dass SQL keine prozedurale Sprache ist. STP's gehören erst seit SQL3 zum Standard, aber praktisch alle grossen DBS unterstützen sie schon lange vorher, selbstverständlich immer mit unterschiedlicher Syntax. Genau dies macht die Portabilität zwischen verschiedenen DB's aber sehr schwierig.

Kreieren einer Stored Procedure

Eine Stored Procedure ist ein DB-Objekt und wird wie ein solches kreiert:

```
CREATE PROCEDURE Procedurname ( @param1 int , @param2 varchar(20) )
AS
```

```
begin transaction
    beliebig viele SQL-Statements
commit transaction
Grant execute on Procedurname to public
```

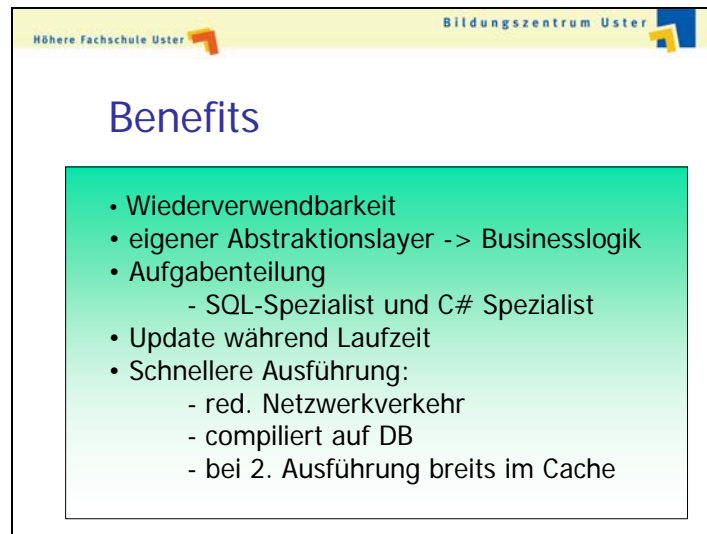
Aufruf einer Stored Procedure

Stored Procedures werden wie Funktionen oder Prozeduren in anderen Programmiersprachen aufgerufen:

```
returnwert = Procedurname param1, param2
```

Features von Stored Procedures

- Aufruf mit Parameter
- Gibt Returnwert zurück
- Benutzung von lokalen Variablen
- if then else Statement
- While – Schleife mit Cursor
- kann Result Set zurück geben
- Benutzung von temporären Tabellen
- Aufruf von anderen Stored Procedures



Höhere Fachschule Uster Bildungszentrum Uster

Benefits

- Wiederverwendbarkeit
- eigener Abstraktionslayer -> Businesslogik
- Aufgabenteilung
 - SQL-Spezialist und C# Spezialist
- Update während Laufzeit
- Schnellere Ausführung:
 - red. Netzwerkverkehr
 - compiliert auf DB
 - bei 2. Ausführung bereits im Cache

Wiederverwendbarkeit

Stored Proc bieten alle Vorteile von Subroutinen in anderen Programmiersprachen: einmal programmieren, vielfach verwenden. Die gemeinsame Nutzung durch mehrere Anwendungen gewährleistet Vereinheitlichung.

Businesslogik

In gespeicherten Procedures lassen sich Geschäftsfunktionen zusammenfassen. Die so zusammengefassten Geschäftsregeln oder –richtlinien können an einem Ort geändert werden. Sämtliche Clients können auf dieselben Stored Proc zugreifen, wodurch bei Änderungen Einheitlichkeit gewährleistet wird. Die Businesslogik ist so in einem eigenen Layer und nicht irgendwo versteckt auf unterschiedlichsten Applikationen auf Client-Seite. Die Applikationen sollten idealerweise nur noch die Daten in Masken, Listen, etc. darstellen und keine eigene Logik enthalten.

Aufgabenteilung

Spezialisten, die alle Technologien im Griff haben und zudem noch die Welt der Anwender verstehen, sind schwer zu finden. Durch die Layerung von SQL auf Stored Proc und C# in den Anwendungen ergibt sich eine klare Schnittstelle; so wird nicht alles mit allem vermischt. SQL- und C# Spezialisten können sich somit auf ihr Layer konzentrieren, müssen sich aber über die Schnittstelle einig sein.

Update während Laufzeit

Stored Proc können während der Laufzeit – also wenn Dutzende von Clients aktiv mit dem Server kommunizieren – auf die DB geladen werden, ohne dass ein Anwender etwas davon merkt.

Schnellere Ausführung

- Wenn ein grosser Stapel von SQL-Anweisungen über ein Netzwerk auf einem Server ausgeführt wird, steht die Anwendung ständig in einem Datenaustausch mit dem Server, wodurch sich schnell eine erhebliche Netzbelastung ergeben kann. Sind mehrere Benutzer mit solchen Anwendungen am Netz, geht die Leistung des Netzwerks schnell mal zurück. Mit einer Stored Proc gehen nur am Schluss der Procedure der Returnwert und wenn notwendig ein Resultset zurück an die Anwendung.
- Eine interaktives SQL-Statement muss vor der Ausführung immer zuerst kompiliert werden. Im Gegensatz dazu liegen die Stored Proc schon in kompilierter Form auf der DB.
- Beim erstmaligen Ausführen einer Stored Proc wird der Abfrageplan im Procedurecache gespeichert, wo sie dann ausgeführt wird. Ein erneutes Ausführen derselben Proc erfolgt schneller, da direkt auf das Cache zurückgegriffen werden kann.

Sicherheitsmechanismen

Es ist möglich, den Benutzern Berechtigungen zur Ausführung einer Stored Proc zu gewähren, selbst wenn sie über keine Berechtigungen verfügen, auf die Tabellen zuzugreifen, auf die sich die gespeicherten Proc's beziehen.

3.5.b Cursor

[illegible]

Idee

Cursor ermöglichen in einem Resultset vorwärts und rückwärts zu navigieren, um Daten Tupel-weise zu verarbeiten. Man kann eine Kombination von lokalen Variablen und einem Cursor verwenden, um jeden Datensatz einzeln zu untersuchen und alle erforderlichen Operationen auszuführen, bevor man zum nächsten Datensatz weitergeht.

Typen

- Lese- vs. Update-Cursor
- feste Durchlaufreihenfolge (sequentieller Cursor) vs. freie Positionierung (Scroll Cursor)

Anwendungen

- Die Arbeitsweise mit einem Cursor ist bei der Einbettung von SQL in eine Applikation (3GL-Sprache), die keine Datenmenge mit unbekannter Grösse auf's Mal verarbeiten kann, zwingend notwendig.
- Mit Hilfe eines Cursors können Aufrufeparameter an eine Stored Procedure übergeben werden, siehe Beispiel unten.

Programmierung

Syntax variiert von DBS zu DBS, hier ein Beispiel mit Transakt SQL:

```

Declare @Name          varchar(30)      -- locale Variable
Declare @Stil          varchar(10)     -- locale Variable

create artist cursor for Select Name, Stil From Künstler

open artist

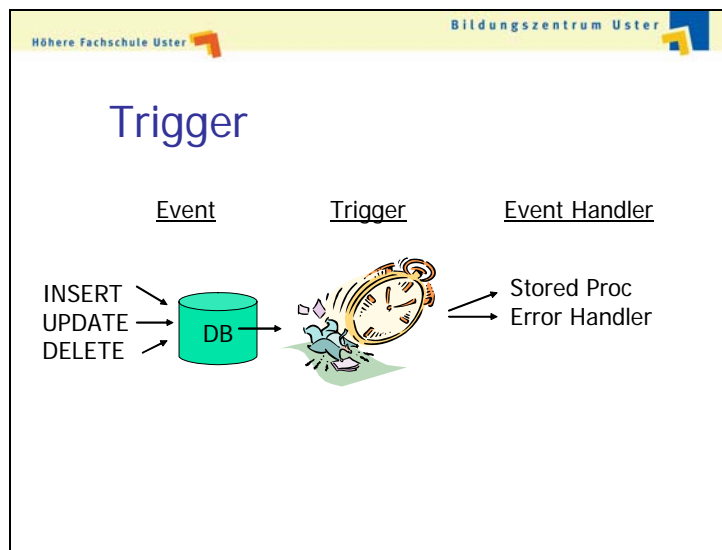
    fetch artist into @Name, @Stil

    while ( @@fetch_status = 0 )
    Begin
        Execute StoredProcedureXYZ @Name, @Stil
        fetch      artist into @Name, @Stil
    End

close artist
deallocate cursor artist

```

3.5.c Trigger



Prinzip

- Ziel und Zweck von Triggern ist die Wahrung von Datenintegrität auf Systemebene.
- Ein Trigger ist eine besondere Form einer Stored Proc, der automatisch ausgeführt wird, sobald der Versuch unternommen wird, Daten in einer Tabelle zu verändern, welche durch einen Trigger geschützt sind.

Merkmale

- Trigger sind an eine Tabelle gebunden.
- Während Stored Proc bewusst vom Anwender (oder Programmierer) aufgerufen werden, werden Trigger durch Ereignisse zwangsweise vom DBS aufgerufen.
- Beim Versuch, Daten in einer Tabelle einzufügen, zu aktualisieren oder zu löschen, wird automatisch der Trigger ausgelöst, wenn für diese bestimmte Aktion ein Trigger für die Tabelle definiert wurde.
- Ein Trigger kann nicht umgangen werden.
- Ein Trigger ist ein DB-Objekt.

```
CREATE TRIGGER TriggerName ON TabellenName FOR Update
AS
    Update      TabellenName
    Set         date  = getdate()
              name = get_suser()
```

Trigger vs Constraints

- Trigger sollten z.B. dann eingesetzt werden, wenn die benötigte Funktionalität nicht mit Hilfe von Constraints erzielt werden kann.
- Der wichtigste Vorteil von Triggern gegenüber Constraints besteht darin, dass sie komplexe Verarbeitungslogiken enthalten können.
- Wenn für Triggertabellen Constraints vorliegen, werden diese zuerst überprüft. Werden Constraints verletzt, wird der Trigger nicht ausgeführt, da die Insert-, Update- oder Delete Anweisung abgebrochen wird.

Einsatzmöglichkeiten

- Auslösen von Fehlermeldungen an den Benutzer.
- Automatisches Einfügen von Systemvariablen (Zeit, Benutzer, ...).
- Historisierung von Veränderungen: wer hat wann welche Änderungen gemacht.

3.6 Übungen

3.6.a Team, SQL Abfragen

gegeben ist folgendes rel. DB-Modell

Team

Nummer*	Name	Ort	Manager
12	Dodgers	Los Angeles	Wilson
15	Glants	San Francisco	Johnson
20	Yankees	New York	Simpson
24	Tigers	Detroit	Corbin

Berufspraxis

ID*	TeamNummer	TrainerName	Typ	Jahre
1	12	Adams	High School Trainer	5
2	12	Adams	College Trainer	10
3	12	Baker	College Trainer	3
4	12	Baker	Japanische Liga Trainer	2
5	12	Baker	Unterliga Trainer	4
6	15	Victor	College Trainer	15
7	24	Dooley	Unterliga Trainer	12

Schläger

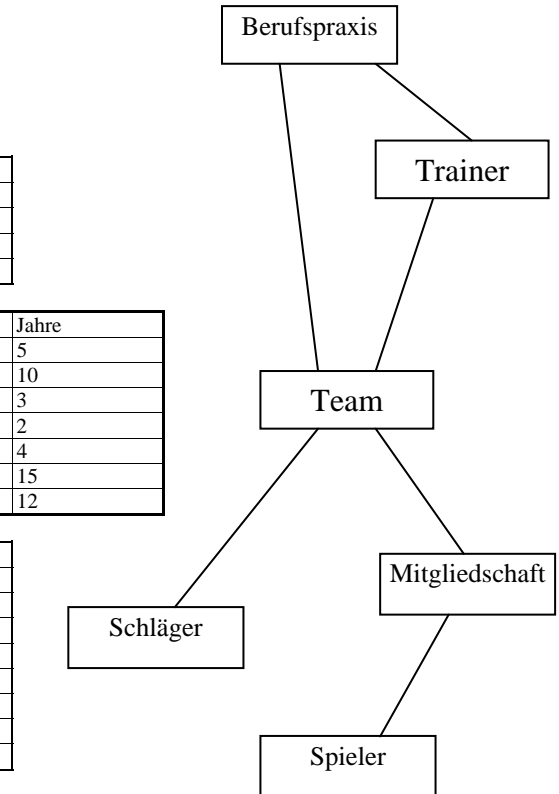
ID*	TeamNummer	Typ	Hersteller
1	12	B01	Acme
2	12	B09	General
3	12	B13	Acme
4	15	B01	Acme
5	20	B03	United
6	20	B04	Modern
7	24	B18	General
8	24	B21	United

Spieler

Nummer*	Name	Alter
358	Stevens	21
523	Doe	32
1131	Johnson	28
1779	Jones	25
2007	Dobbs	27
4280	Cohen	25
4319	Ross	24
5410	Smith	27
6564	Linton	24
8093	Smith	21
8366	Gomez	33

Mitgliedschaft

ID*	SpielerNummer	TeamNummer	Jahre	Durchschnitt
1	358	15	3	0.3
2	358	20	3	0.32
3	523	12	10	0.257
4	1131	20	1	0.283
5	1779	12	1	0.223
6	1779	15	7	0.246
7	1779	24	2	0.24
8	2007	24	3	0.29
9	4280	15	1	0.195
10	4280	20	3	0.227
11	4319	15	4	0.298
12	5410	12	6	0.307
13	6564	20	12	0.31
14	6564	24	3	0.28
15	8093	12	5	0.25
16	8093	20	2	0.24
17	8093	24	8	0.265
18	8366	20	7	0.283



Trainer

Name*	TeamNummer	Telefon
Adams	12	555-1524
Baker	12	555-3690
Taylor	15	555-4820
Jackson	20	555-2444
Victor	24	555-9327
Dooley	24	555-7321

Team, SQL Abfragen

- 1) Die gesamte Tabelle Team
- 2) Datensatz aus Tabelle Team mit Nummer 12
- 3) Liste mit Nummer und Namen aller Teams
- 4) Liste mit Nummer und Namen aller Spieler, die älter als 30 sind.
- 5) Liste mit Teamnummer und Namen aller College Trainer, die mehr als 5 Jahre Berufspraxis aufweisen
- 6) Liste aller Datensätze aus der Tabelle Berufspraxis mit Typ College Trainer oder Berufspraxis grösser 10 Jahre
- 7a) Liste aller Teamnummern der Teams, deren Schläger von Acme ist
- 7b) wie 7a), aber keine mehrfachen Teamnummern
- 8) Liste aller Spieler, mit Alter zwischen 25 und 29, aufsteigend sortiert nach Alter
- 9) Liste aller Schläger, mit Hersteller General oder United
- 10) Liste aller Spieler, deren Namen mit S beginnen
- 11) Anzahl Teams
- 12) Anzahl Spieler, älter als 30
- 13) Anzahl Jahre Berufspraxis von Trainer Adams
- 14) Liste mit Teamnummer, Trainername und Anzahl Jahre Berufspraxis, gruppiert nach Teamnummer, Trainername
- 15) wie 14), aber nur Trainer mit mehr als 10 Jahren Berufspraxis
- 16) Liste alle Trainer inklusive Teams. Sämtliche Informationen in beiden Tabellen sollen dargestellt sein.
- 17) Liste aller Trainer, die ein Team in Detroit trainieren. Darstellung wie in 16)
- 18) Liste aller Trainernamen, die das Team Dodgers trainieren.
- 19) Liste die Trainer der Liga, ihre Telefonnummer und die Namen der Teams, für die sie arbeiten, auf.
- 20) Liste mit Spielernummer und –Name, die bei Dodgers spielen oder gespielt haben, sowie die Anzahl Jahre in diesem Team, inklusive der entsprechenden Durchschnittsleistung.
- 21) Das Alter des ältesten Spielers, der bei den Dodgers spielt oder gespielt hat.
- 22) Liste aller Trainer mit Namen und Telefon, die das Team Dodgers trainieren.
- 23) Liste mit Nummer, Name und Alter aller Spieler, die älter sind als der Durchschnitt.
- 24) Spieler mit Nummer und Name, die 5 oder mehr Jahre in einem Team gespielt haben.

Zusatz

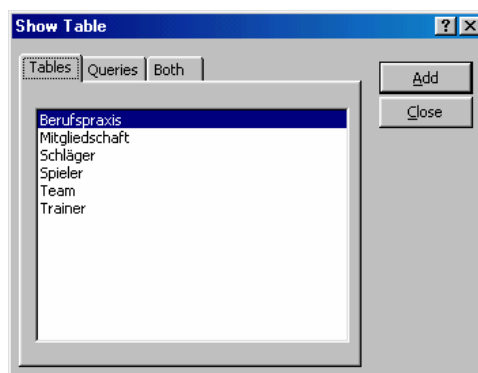
Welche Punkte sind in den Tabellen vom Beispiel Team schlecht gelöst ?

SQL Statements mit Access ausführen

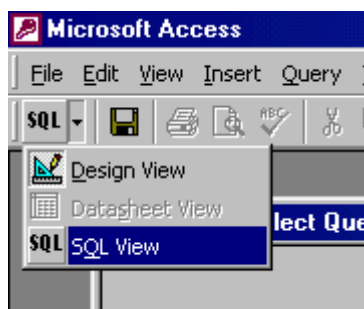
1. Doppelklicken Sie im Explorer auf 'Team.mdb'. Access wird so gestartet und die entsprechende Datenbank geladen.
2. Selektieren Sie im Hauptmenü 'Objects' den Eintrag **Queries**. Dort wählen Sie **Create query in Design view**.



3. Den Dialog **Show Table** schliessen Sie mit **Close**.



4. Im Toolbar Button **SQL** wählen Sie **SQL View**.



5. Im neu aufgegangenen Fenster können Sie Ihr SQL Statement formulieren: z.B.
SELECT * from Spieler
6. Um dieses Statement auszuführen klicken Sie auf das **rote Ausrufesymbol !** in der Toolbar. Sie erhalten so das **Result Set** in einem neuen Fenster.

Team Lösungen

- 1) SELECT *
 FROM Team

Nummer	Name	Ort	Manager
12	Dodgers	Los Angeles	Wilson
15	Glants	San Francisco	Johnson
20	Yankees	New York	Simpson
24	Tigers	Detroit	Corbin

- 2) SELECT *
 FROM Team
 WHERE Nummer = 12

Nummer	Name	Ort	Manager
12	Dodgers	Los Angeles	Wilson

- 3) SELECT Nummer, Name
 FROM Team

Nummer	Name
12	Dodgers
15	Glants
20	Yankees
24	Tigers

- 4) SELECT Nummer , Name
 FROM Spieler
 WHERE Alter > 30

Nummer	Name
523	Doe
8366	Gomez

- 5) SELECT TeamNummer, TrainerName
 FROM Berufspraxis
 WHERE Typ = "College Trainer"
 AND Jahre > 5

TeamNummer	TrainerName
12	Adams
15	Victor

- 6) SELECT *
 FROM Berufspraxis
 WHERE Typ = "College Trainer"
 OR Jahre > 10

ID	TeamNummer	TrainerName	Typ	Jahre
2	12	Adams	College Trainer	10
3	12	Baker	College Trainer	3
6	15	Taylor	College Trainer	15
7	24	Victor	Unterliga Trainer	12

- 7)

SELECT TeamNummer FROM Schläger

SELECT DISTINCT TeamNummer FROM Schläger

WHERE Hersteller = "Acme"
TeamNummer
12
12
15

WHERE Hersteller = "Acme"
TeamNummer
12
15

- 8) SELECT *
 FROM Spieler
 WHERE Alter BETWEEN 25 AND 29
 ORDER BY Alter

Nummer	Name	Alter
4280	Cohen	25
1779	Jones	25
5410	Smith	27
2007	Dobbs	27
1131	Johnson	28

- 9) SELECT *
 FROM Schläger
 WHERE Hersteller IN ("General", "United")

ID	TeamNummer	Typ	Hersteller
2	12	B09	General
5	20	B03	United
7	24	B18	General
8	24	B21	United

- 10) SELECT *
 FROM Spieler
 WHERE Name LIKE "S*"

Nummer	Name	Alter
358	Stevens	21
5410	Smith	27
8093	Smith	21

- 11) SELECT COUNT (*) as `Anzahl`
 FROM Team

Anzahl
4

- 12) SELECT COUNT (*) as `Anzahl über 30`
 FROM Spieler
 WHERE Alter > 30

Anzahl über 30
2

- 13) SELECT SUM(Jahre)
 FROM Berufspraxis
 WHERE TrainerName = "Adams"

Expr1000
15

- 14) SELECT TeamNummer, TrainerName, SUM(Jahre)
FROM Berufspraxis
GROUP BY TeamNummer, TrainerName

TeamNummer	TrainerName	Expr1002
12	Adams	15
12	Baker	9
15	Taylor	15
24	Victor	12

- 15) SELECT TeamNummer, TrainerName, SUM(Jahre) AS `Praxis`
FROM Berufspraxis
GROUP BY TeamNummer, TrainerName
HAVING SUM(Jahre) > 10

TeamNummer	TrainerName	Praxis
12	Adams	15
15	Taylor	15
24	Victor	12

- 16) SELECT Trainer.*, Team.*
FROM Trainer, Team
WHERE Team.Nummer = Trainer.TeamNummer

Trainer.Name	Trainer.TeamNr	Telefon	Nummer	Team.Name	Ort	Manager
Adams	12	555-1524	12	Dodgers	Los Angeles	Wilson
Baker	12	555-3690	12	Dodgers	Los Angeles	Wilson
Taylor	15	555-4820	15	Glants	San Francisco	Johnson
Jackson	20	555-2444	20	Yankees	New York	Simpson
Victor	24	555-9327	24	Tigers	Detroit	Corbin
Dooley	24	555-7321	24	Tigers	Detroit	Corbin

- 17) SELECT Tr.*, Te.*
FROM Trainer Tr, Team Te
WHERE Te.Nummer = Tr.TeamNummer
AND Ort = "Detroit"

Trainer.Name	Trainer.TeamNr	Telefon	Nummer	Team.Name	Ort	Manager
Victor	24	555-9327	24	Tigers	Detroit	Corbin
Dooley	24	555-7321	24	Tigers	Detroit	Corbin

- 18) SELECT t1.Name
FROM Trainer t1, Team t2
WHERE t2.Nummer = t1.TeamNummer
AND t2.Name = "Dodgers"

Trainer.Name
Adams
Baker

- 19) SELECT Trainer.Name, Trainer.Telefon, Team.Name
FROM Trainer, Team
WHERE Team.Nummer = Trainer.TeamNummer

Trainer.Name	Trainer.Telefon	Team.Name
Adams	555-1524	Dodgers
Baker	555-3690	Dodgers
Taylor	555-4820	Glants

Jackson	555-2444	Yankees
Victor	555-9327	Tigers
Dooley	555-7321	Tigers

- 20) SELECT S.Nummer, S.Name, M.Jahre, M.Durchschnitt
 FROM Spieler S, Mitgliedschaft M, Team T
 WHERE S.Nummer = M.SpielerNummer
 AND M.TeamNummer = T.Nummer
 AND T.Name = "Dodgers"

Spieler.Nummer	Spieler.Name	Jahre	Durchschnitt
523	Doe	10	0.257
1779	Jones	1	0.223
5410	Smith	6	0.307
8093	Smith	5	0.25

- 21) SELECT MAX (Alter) as `Alter`
 FROM Spieler S, Mitgliedschaft M, Team T
 WHERE S.Nummer = M.SpielerNummer
 AND M.TeamNummer = T.Nummer
 AND T.Name = "Dodgers"

Alter
32

- 22) SELECT Name, Telefon
 FROM Trainer
 WHERE TeamNummer IN (SELECT Nummer FROM Team WHERE Name = "Dodgers")

Name	Telefon
Adams	555-1524
Baker	555-3690

- 23) SELECT *
 FROM Spieler
 WHERE Alter > (SELECT AVG(Alter) FROM Spieler)

Nummer	Name	Alter
523	Doe	32
1131	Johnson	28
2007	Dobbs	27
5410	Smith	27
8366	Gomez	33

- 24) SELECT Nummer, Name
 FROM Spieler
 WHERE Nummer IN (SELECT SpielerNummer FROM Mitgliedschaft WHERE Jahre >= 5)

Nummer	Name
523	Doe
1779	Jones
5410	Smith
6564	Linton
8093	Smith
8366	Gomez

3.6.b Firma, SQL Abfragen

1. Liste mit Name und Vorname aller Personen
2. Die gesamte Tabelle der Personen
3. Die gesamte Tabelle der Adressen
4. Alle Personen mit Namen „Baumann“
5. Alle Frauen mit Namen und Vornamen
6. Alle Personen mit Geschlecht welche den Jahrgang älter als 1970 haben
7. Alle Firmen welche im Segment „Maschinenbau“ tätig sind
8. Adressen welche in Zürich am Paradeplatz zu finden sind
9. Alle Adressen, die in der Postleitzahl mit 8 beginnen (Achtung PLZ ist ein String)
10. Die Telefonnummern der Firma Alfa Laval
11. Eine Telefonliste aller Firmen mit all ihren Nummern
12. Eine reduzierte Liste der Natel Nummern oder Email Adressen von Firmen
13. Telefonnummern von Firmen, welche im Maschinenbau tätig sind
14. Alle Segmente, von welchen Firmen in der Datenbank aufgenommen sind
15. Jedes Segment von Aufgabe 14 soll nur einmal im Resultat erscheinen
16. Personen, welche Jahrgang zwischen 65 und 70 aufweisen
17. Die Anzahl der erfassten Telefonnummern
18. Der durchschnittliche Jahrgang der Personen mit gleichem Nachnamen
19. Nur die Familiennamen, bei welchen obiger Durchschnitt älter ist als Jahrgang 65
20. Die Adresse des Herrn Gisi
21. Die Telefonnummer von Herrn Gisi
22. Das Durchschnittsalter der Mitarbeiter der Firma Alfa Laval
23. Die Mutter von Herr Roman Baumann
24. Der Grossvater mütterlicherseits von Herr Roman Baumann
25. Adressen, an denen mehr als zwei Personen wohnen
26. Die Geschäftsnummer von Herr Christoph Baumann

Telefon

ID	Nummer	Type	FirmaID
1	01'804'66'00	Telefon	1
2	01'304'67'11	Telefon	2
3	01'930'59'50	Natel	3
4	01'701'21'39	Telefon	4
5	031'781'20'53	Telefon	5
8	01'804'67'00	Fax	1
9	042'312'66'23	Telefon	12
11	01'920'61'81	Fax	3
12	AlfaLaval@cyberlink.ch	email	1
13	062'351'22'79	Telefon	7
14	01'41'92'02	Telefon	6
15	042'344'23'89	Fax	12
16	01'341'34'46	Telefon	8
17	052'633'18'87	Telefon	0
18	01'741'08'63	Telefon	9
19	01'341'69'23	Telefon	10
20	01'791'10'83	Fax	3
21	056'839'18'10	Telefon	0
22	077'244'84'41	Natel	0
23	077'352'29'69	Natel	0
24	032'947'13'42	Telefon	0
25	01'462'59'43	Telefon	11
26	071'245'90'80	Fax	0
27	055'740'18'44	Telefon	0
28	01'371'05'52	Telefon	13
29	032'640'39'50	Telefon	0
30	058'640'84'58	Fax	0
31	081'325'16'28	Telefon	0
32	056'51'24'64	Fax	0
33	056'422'93'78	Telefon	0
34	Admin@abb.com	email	4
42	032'41'59'33	Telefon	0

Adresse

ID	Adresse	PLZ	Ort
5	Scherzerstr. 7	5116	Schinznach-Bad
6	Perronweg 8	8855	Wangen
7	Im Grüntal 13	8405	Winterthur
8	Uttenberg	8934	Knonau
9	Klausfeld 3	6037	Root
10	Oberhardstr. 28	5413	Birmenstorf
11	Kaltbreiterstrasse 99	8003	Zürich
12	Sonnenbergstr. 2	4127	Birsfelden
13	Gerenstr. 40	9202	Gossau
14	Landstr. 56	4452	Ittingen
15	Möhrlistr. 91	8006	Zürich
16	Rütistrasse 226	4703	Kestenholz
17	Mühlethal 6	3270	Aarberg
18	äussere Stammerau 13	8500	Frauenfeld
19	Zürcherstr. 19	8154	Oberglatt
20	Via San Gian 26	7500	St. Moritz
21	Spinnerstr. 29	8640	Rapperswil
22	Alte Stationsstrasse 22	8154	Oberglatt
23	Landstr. 73	8450	Andelfingen
25	Oberfeldstrasse 20	8302	Kloten
26	Albisgüetli	8000	Zürich
27	Brown-Boweristr. 1	8000	Zürich
28	Dufourstr.	3001	Bern
29	Industrie 21/b	8040	Altstetten
30	Zentralquartier	5030	Aarau
31	Bahnhofstr. 28	8153	Glattbrugg
32	Tiefenbrunnen 12	8057	Zürich
33	Brunau	8091	Zürich
34	Baarerstr. 112	6000	Zug
35	Zentrum Glatt	8150	Wallisellen
36	Paradeplatz	8000	Zürich

Firma

ID	Name	Segment	AdresseID
1	Alfa Laval	Automation	25
2	Tetra Pak	Verpackung	25
3	SKA	Bank	26
4	ABB	Maschinenbau	27
5	Ascom Hasler	Rüstung	28
6	Sulzer	Maschinenbau	29
7	Holderbank	Zement	30
8	Messerli	Druckerei	31
9	SBG	Bank	36
10	Elektrowatt	Ingenieurbüro	32
11	Locher	Bau	33
12	Landis & Gyr	Gebäudetechnik	34
13	Kuoni	Reisebüro	35

Beziehung

ID	Beziehungstyp	Person1ID	Person2ID	ID	FirmaID	PersonID
1	Ehepaar	1	4	1	1	5
2	Elternteil	1	19	2	1	20
3	Elternteil	4	19	3	1	1
4	Paar	15	22	4	5	20
5	Elternteil	10	1	5	5	16
6	Elternteil	29	16	6	5	4

PersonFirma

Person

ID	Anrede	Name	Vorname	Geburtsjahr	Geschlecht
1	Frau	Baumann	Renate	65	f
3	Herr	Brandenberg	Lukas	93	m
4	Herr	Baumann	Christoph	63	m
5	Herr	Frischknecht	Markus	69	m
6	Herr	Geiser	Christof	67	m
7	Herr	Gisi	Stefan	42	m
8	Herr	Guglielmetti	Didier	65	m
9	Herr	Gysler	René	58	m
10	Herr	Huber	Rene	39	m
11	Herr	Imhof	Felix	62	m
12	Herr	Irniger	Klaus	61	m
13	Herr	Kissling	Urs	64	m
14	Herr	Kohler	Manuel	67	m
15	Herr	Krucker	Beat	72	m
16	Frau	Meier	Esther	69	f
17	Herr	Nydegger	HansJürg	67	m
18	Herr	Peterka	Boris	67	m
19	Herr	Baumann	Roman	85	m
20	Herr	Rüsch	Markus	35	m
21	Herr	Scherrer	Fabian	55	m
22	Frau	Signer	Nicole	69	f
29	Herr	Vollmer	Patrick	31	m

PersonTelefon

ID	PersonID	TelefonID
1	1	24
2	4	24
3	19	24
4	4	5
5	5	17
6	6	21
7	14	21
8	17	21
9	6	4
10	14	4
11	17	18
12	7	22
13	8	23
14	9	27
15	21	31
16	9	16
17	21	16
18	20	42
19	21	33
20	22	31
21	21	32
22	29	29

PersonAdresse

ID	PersonID	AdresseID
1	1	16
2	3	18
3	4	16
4	5	18
5	6	10
6	7	11
7	7	7
8	8	23
9	9	21
10	10	8
11	11	19
12	12	22
13	12	9
14	13	5
15	14	10
16	15	20
17	16	6
18	17	10
19	18	15
20	19	16
21	20	17
22	21	21
23	22	20
24	29	6

Firma Lösungen

- 1) SELECT Name, Vorname
 FROM Person

Name	Vorname
Baumann	Renate
Brandenberg	Lukas
Baumann	Christoph
Frischknecht	Markus
Geiser	Christof
...	

- 2) SELECT *
 FROM Person

- 3) SELECT *
 FROM Adresse

- 4) SELECT *
 FROM Person
 WHERE Name = "Baumann"

- 5) SELECT Name, Vorname
 FROM Person
 WHERE Geschlecht = "f"

- 6) SELECT Name, Vorname, Geschlecht
 FROM Person
 WHERE Geburtsjahr < 70

- 7) SELECT *
 FROM Firma
 WHERE Segment = "Maschinenbau"

- 8) SELECT *
 FROM Adressen
 WHERE Ort = "Zürich"
 AND Adresse = "Paradeplatz"

- 9) SELECT Adresse, plz, Ort
 FROM Adresse
 WHERE plz LIKE "8*"

Adresse	plz	Ort
Perronweg 8	8855	Wangen
Im Grüntal 13	8405	Winterthur
Uttenberg	8934	Knonau
...		

- 10) SELECT T.Nummer, T.Type
 FROM Telefon T, Firma F
 WHERE T.FirmaID = F.ID

AND F.Name = "Alfa Laval"

Nummer	Type
01'804'66'00	Telefon
01'804'67'00	Fax
AlfaLaval@cyberlink.ch	email

- 11) SELECT F.Name, T.Nummer, T.Type
FROM Telefon T, Firma F
WHERE T.FirmaID = F.ID
- 12) SELECT Nummer, Name
FROM Telefon, Firma
WHERE Telefon.FirmaID = Firma.ID
AND Telefon.Type IN ("Natel", "email")
- 13) SELECT Nummer, Name
FROM Telefon, Firma
WHERE Telefon.FirmaID = Firma.ID
AND Segment = "Maschinenbau"
- 14) SELECT Segment
FROM Firma
- 15) SELECT DISTINCT Segment
FROM Firma
- 16) SELECT Name, Vorname
FROM Person
WHERE Geburtsjahr BETWEEN 65 AND 70
- 17) SELECT COUNT (*)
FROM Telefon
- 18) SELECT Name, AVG(Geburtsjahr)
FROM Person
GROUP BY Name
- 19) SELECT Name, AVG(Geburtsjahr)
FROM Person
GROUP BY Name
HAVING AVG(Geburtsjahr) < 65
- 20) SELECT A.Adresse, A.plz, A.Ort
FROM Person P, PersonAdresse PA, Adresse A
WHERE PA.PersonID = P.ID
AND PA.AdresseID = A.ID
AND P.Name = "Gisi"

Adresse	plz	Ort
Kaltbreiterstrasse 99	8003	Zürich
Im Grüntal 13	8405	Winterthur

- 21) SELECT T.Nummer, T.Type
 FROM Telefon T, PersonTelefon PT, Person P
 WHERE PT.TelefonID = T.ID
 AND PT.PersonID = P.ID
 AND P.Name = "Gisi"

Nummer	Type
077'244'84'41	Natel

- 22) SELECT 107 - Sum(Geburtsjahr) / Count(Geburtsjahr) AS `Durchschnittsalter`
 FROM Person P, PersonFirma PF, Firma F
 WHERE PF.PersonID = P.ID
 AND PF.FirmaID = F.ID
 AND F.Name = "Alfa Laval"

Durchschnittsalter
50.6666666666667

- 23) SELECT Mutter.Name, Mutter.Vorname
 FROM Person as Mutter, Person as Sohn, Beziehung B
 WHERE Mutter.ID = B.Person1ID
 AND Sohn.ID = B.Person2ID
 AND Sohn.Name = "Baumann"
 AND Sohn.Vorname = "Roman"
 AND B.BeziehungsTyp = "Elternteil"
 AND Mutter.Geschlecht = "f"

Name	Vorname
Baumann	Renate

- 24) SELECT Grossvater.Name, Grossvater.Vorname
 FROM Person as Mutter, Person as Sohn, Beziehung as Eltern, Beziehung as GrossEltern,
 Person as Grossvater
 WHERE Mutter.ID = Eltern.Person1ID
 AND Sohn.ID = Eltern.Person2ID
 AND Sohn.Name = "Baumann"
 AND Sohn.Vorname = "Roma"
 AND Eltern.BeziehungsTyp = "Elternteil"
 AND Mutter.Geschlecht = "f"
 AND GrossEltern.Person2ID = Mutter.ID
 AND GrossEltern.Person1ID = Grossvater.ID
 AND Grossvater.Geschlecht = "m"

Name	Vorname
Huber	Rene

- 25) SELECT plz, Ort
 FROM Adresse, PersonAdresse
 WHERE Adresse.ID = AdresseID
 GROUP BY plz, Ort
 HAVING COUNT (PersonID) > 2

oder:

- SELECT Adresse, plz, Ort
 FROM Adresse
 WHERE Adresse.ID IN (SELECT AdresseID
 FROM PersonAdresse

GROUP BY AdresseID
HAVING COUNT(PersonID) > 2)

plz	Ort
4703	Kestenholz
5413	Birmenstorf

26) Select T.Nummer, T.Type
 From Telefon T, Person P, PersonFirma PF, Firma F
 Where P.Name = "Baumann"
 and P.Vorname = "Christoph"
 and PF.PersonID = P.ID
 and PF.FirmaID = F.ID
 and T.firmaID = F.ID

Nummer	Type
031'781'20'53	Telefon

3.6.c DDL und DML

- 1) Kreieren Sie mit SQL-Statements eine Tabelle Blume mit den Attributen:
ID, Name, Preis
- 2) Fügen Sie zwei Datensätze zu mit den Werten:
10, Nelke, 2.35
11, Rose, 4.50
- 3) Überprüfen Sie diese Einträge mit einem SELECT-Statement.
- 4) Erhöhen Sie den Preis der Rose um 10%.
- 5) Überprüfen Sie den Update in 4) mit einem SELECT-Statement.
- 6) Löschen Sie den Datensatz mit der Nelke.
- 7) Entfernen Sie die gesamte Tabelle Blume, inklusive Metadaten.

3.6.d Constraints

1. Welche Behauptung ist wahr ?
 - a) Constraints sind Einschränkungen, die vom DBS automatisch generiert werden und deren Einhaltung vom DBS auch überwacht wird.
 - b) Constraints sind Nötigungen der DB an den Benutzer.
 - c) Constraints müssen vom Programmierer definiert werden.
 - d) Ein Primary Key ist kein Constraint

2. Welches Statement ist korrekt ?
 - a) Ein Primary Key erlaubt NULL-Werte.
 - b) Ein Unique Constraint erlaubt NULL-Werte.
 - c) Ein Default-Constraint schränkt den Wertebereich eines Attributes ein.
 - d) Maximal 1 Unique Constraint ist pro Tabelle erlaubt.

3. Ordnen Sie folgende Begriffe einander zu.
 - a) Domänen-Integrität
 - b) Partielle-Integrität
 - c) Entitäts-Integrität
 - d) Referentielle-Integrität
 - e) a) und c) und d)
 - 1) Default-Constraint gehört in diese Gruppe.
a) b) c) d)
 - 2) Wird durch den Primary-Key garantiert.
a) b) c) d)
 - 3) Auf eine Verletzung dieser Integrität kann auf verschiedene Art und Weise reagiert werden.
a) b) c) d)
 - 4) Können Bestandteil des Create Table Befehls sein.
a) b) c) d) e)

4. Welche Aussagen sind korrekt ?

ja	nein	
<input type="checkbox"/>	<input type="checkbox"/>	Nach einem DELETE kann kein DROP TABLE mehr ausgeführt werden.
<input type="checkbox"/>	<input type="checkbox"/>	Nach einem DROP TABLE kann kein INSERT mehr ausgeführt werden.
<input type="checkbox"/>	<input type="checkbox"/>	Nach einem DELETE kann kein INSERT mehr ausgeführt werden.
<input type="checkbox"/>	<input type="checkbox"/>	Nach einem DROP TABLE ist die Tabelle leer.
<input type="checkbox"/>	<input type="checkbox"/>	Nach einem DELETE ohne WHERE-Klausel kann kein DROP Table mehr ausgeführt werden.

- 5: Was wären die Konsequenzen, wenn keine SQL-Constraints zur Verfügung ständen ?

3.6.e Index, View

- 1) Ein Index garantiert in jedem Fall eine schnellere Antwortzeit.
☐ ja ☐ nein
☐ o ☐ o
- 2) Bei grossen Tabellen sollten auf jeden Fall alle Spalten einen Index haben.
☐ ja ☐ nein
☐ o ☐ o
- 3) Kleine Tabellen sollten höchstens 2 Indices haben.
☐ ja ☐ nein
☐ o ☐ o
- 4) Das DBS entscheidet selber, ob via Tablescan oder via Index zugegriffen werden soll.
☐ ja ☐ nein
☐ o ☐ o
- 5) Die View gehört je nach Sichtweise zur konzeptionellen oder externen Ebene.
☐ ja ☐ nein
☐ o ☐ o
- 6) Eine View kann als virtuelle Tabelle betrachtet werden.
☐ ja ☐ nein
☐ o ☐ o
- 7) Eine View hat Daten abgespeichert, die jederzeit eingesehen werden können.
☐ ja ☐ nein
☐ o ☐ o
- 8) Mit Hilfe von Views können für verschiedene Benutzer(gruppen) gezielte Spalteninformationen ein- oder ausgeblendet werden.
☐ ja ☐ nein
☐ o ☐ o
- 9) Eine View speichert Daten, analog zu einer Tabelle.
☐ ja ☐ nein
☐ o ☐ o
- 10) Access bietet ein grafisch orientiertes Tool an, um SQL-Statements auf Mausklick hin zu erstellen. Warum sollte ich meine Zeit mit dem manuellen Schreiben von SQL verschwenden?

3.6.f Transaktion

1. Was bringt das Transaktions-Konzept in einer Single-User-Umgebung ?

ja nein

☐ ☐ nichts

☐ ☐ Performance-Verlust

☐ ☐ Prestige

☐ ☐ Recovery-Verfahren nach einem Absturz
2. Atomarität bedeutet, dass eine Transaktion entweder ganz oder gar nicht ausgeführt wird.

ja nein

☐ ☐
3. Während eine Transaktion läuft, können Konsistenzbedingungen verletzt sein.

ja nein

☐ ☐
4. Das Transaktions-Konzept ist Voraussetzung für das Recovery nach einem Absturz.

ja nein

☐ ☐
5. Das Transaktions-Konzept ist Voraussetzung, um in einem Mehrbenutzerbetrieb die Parallelität zu serialisieren.

ja nein

☐ ☐
6. Nach einem Commit kann ein Rollback ausgeführt werden.

ja nein

☐ ☐
7. Das Pessimistische Verfahren bei Transaktionen ist einfacher als das Optimistische.

ja nein

☐ ☐
8. Das Transaction-Log-Konzept entspricht einem Protokollierungs-Verfahren.

ja nein

☐ ☐
9. Nach einem System-Absturz wird beim Booten eine abgebrochene Transaktion ...
 - a) fertig ausgeführt -> Rollforward
 - b) zurückgesetzt -> Rollback
 - c) nichts unternommen

10. Erklären Sie mit Stichworten die Funktionsweise eines Rollbacks.

3.6.g Benutzerrechte, Sicherheitsmechanismen

1) Wer ist der Besitzer einer Tabelle ?

ja nein

- ☐ ☐ Derjenige, der einer Tabelle zuerst einen Datensatz einfügt.
- ☐ ☐ Derjenige, der die Session gestartet hat.
- ☐ ☐ Derjenige, der die Tabelle kreiert hat.
- ☐ ☐ Derjenige, der die höchste Berechtigung hat.
- ☐ ☐ Jeder, der die Tabelle benutzt.

2) Wer darf in einem positiven Schutzsystem ein DB-Objekt ändern ?

ja nein

- ☐ ☐ nur der Besitzer
- ☐ ☐ jeder mit einem gültigen Passwort
- ☐ ☐ jeder mit einer Änderungs-Berechtigung
- ☐ ☐ derjenige, der das DB-Objekt kreiert hat
- ☐ ☐ DBA

3) Sie möchten eine Leseberechtigung für einzelne Spalten einer Tabelle einer Gruppe von Leuten erteilen. Berechtigungen können Sie aber nur für eine ganze Tabelle (alle Spalten) vergeben. Wie gehen Sie vor ?

4) Zählen Sie alle Ihnen bekannten Sicherheitsmechanismen eines Datenbanksystems auf.

5) Sind folgende Statements korrekt ?

ja nein

- ☐ ☐ GRANT ALL ON Products To PUBLIC
- ☐ ☐ REVOKE ALL ON Categories FROM Ivan Restrict, Cascade
- ☐ ☐ GRANT Update, Insert, Delete ON Salary TO Public

6) Mit welchen Mechanismen können Sie folgende Szenarien verhindern ?

- a. Stromausfall:
- b. Datenanomalien:
- c. Daten-Missbrauch:
- d. Schlechtes Datenmodell:
- e. Harddisk Crash:
- f. Panik bei Serverausfall:

3.6.h Stored Procedures, Triggers

- 1) Folgende Aussagen gelten für Stored Procedures.

ja nein

- ☐ ☐ STP sind DB-Objekte.
- ☐ ☐ STP entsprechen Prozeduren in andern Programmiersprachen.
- ☐ ☐ STP erschweren die Portabilität zwischen DBS.
- ☐ ☐ In STP's kann sich die Business Logik widerspiegeln.
- ☐ ☐ mehrere STP können in Komponenten zusammengefasst werden (DLL, exe, ActiveX)

- 2) Folgende Aussagen gelten für Triggers.

ja nein

- ☐ ☐ Triggers sind DB-Objekte.
- ☐ ☐ Ein Trigger wird vom Programmierer programmiert und aufgerufen.
- ☐ ☐ Triggers sind spezielle STP, die aber vom System aufgerufen werden.
- ☐ ☐ Mehrere Triggers pro Tabelle sind möglich.

- 3) Sie möchten Benutzern in der Lohnbuchhaltung das Einfügen, Aktualisieren und Löschen von Daten in der Tabelle Payroll ermöglichen. Allerdings möchten Sie nicht, dass die Benutzer direkt (via Insert, Update, Delete) auf diese Tabelle zugreifen können. Wie setzen Sie diese Vorgaben um ?

- 4) Sind folgende Statements korrekt ?

ja nein

- ☐ ☐ Trigger werden zur Wahrnehmung der Datenintegrität auf Systemebene eingesetzt.
- ☐ ☐ Ein wichtiges Merkmal von Triggern ist, dass sie komplexe Verarbeitungslogiken enthalten können.
- ☐ ☐ Trigger sollten nur eingesetzt werden, wenn die benötigte Funktionalität nicht mit Constraints abgedeckt werden können.
- ☐ ☐ Trigger sollten keine Result Sets zurückgeben.

- 5) Wie gehe ich vor, wenn ich nachweisen möchte, wer wann welche Änderung via Applikation in einer Tabelle gemacht hat ? Änderungen, die interaktiv mit SQL-Statements abgesetzt wurden, sollen auch erfasst werden.

3.7 Übungen mit MS SQL Server



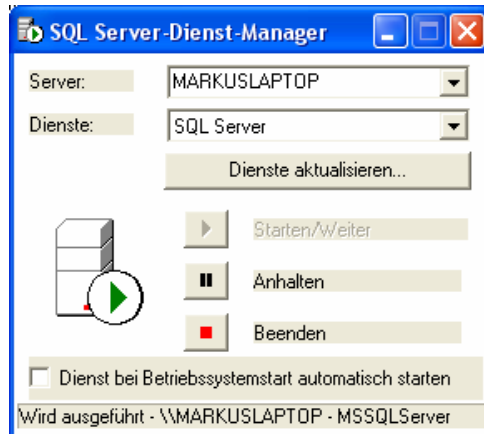
So installieren Sie die 120-Tage-Evaluation-Edition von MS SQL Server 2000

1. Legen Sie die CD-ROM in das CD-ROM-Laufwerk ein.
2. Starten Sie cd-drive:\Server2k\autorun.exe
3. Wählen Sie im Hauptmenü „**SQL-Server 2000-Komponenten (C)**“
4. Im nächsten Menü selektieren Sie „**Datenbankserver – installieren**“
5. Einen allfälligen Hinweis „Die Serverkomponente ... wird nicht unterstützt. Nur Clientkomponenten sind verfügbar.“ bestätigen Sie mit OK. Hier müssen Sie die Installation abbrechen. Installieren Sie statt dessen die Vorgängerversion 7.0
6. Computer Name = **Lokaler Computer**
7. Installationsauswahl = Eine neue Instanz von SQL Server installieren.
8. Installations Definition = Server- und Clienttools
9. Instanzname = Standard
10. Setup-Typ = Standard
11. Dienstkonten = Dasselbe Konto für jeden Dienst verwenden.
Diensteinstellungen = **Konto lokales System verwenden.**
12. Authentifizierungsmodus = Windows-Authentifizierung
13. Restarten Sie den Computer.

3.7.a So verwenden Sie den SQL Server Query Analyzer

In dieser Übung machen Sie sich mit den wichtigsten Funktionen des Query Analyzers vertraut.

1. Doppelklicken Sie das Symbol von **SQL Server-Dienst-Manager** in der Taskleiste oder via Start / Programme / Microsoft SQL Server. Ein grosses, rotes Quadrat symbolisiert, dass der Server noch nicht läuft. Klicken Sie in diesem Fall auf 'Starten/Weiter' und warten Sie einige Sekunden bis das grosse Symbol auf ein grünes Dreieck wechselt.



2. Öffnen Sie den **Query Analyzer** über Start / Programme / Microsoft SQL Server / Query Analyzer. Stellen Sie eine Verbindung zum SQL Server her mit den Default-Einstellungen.



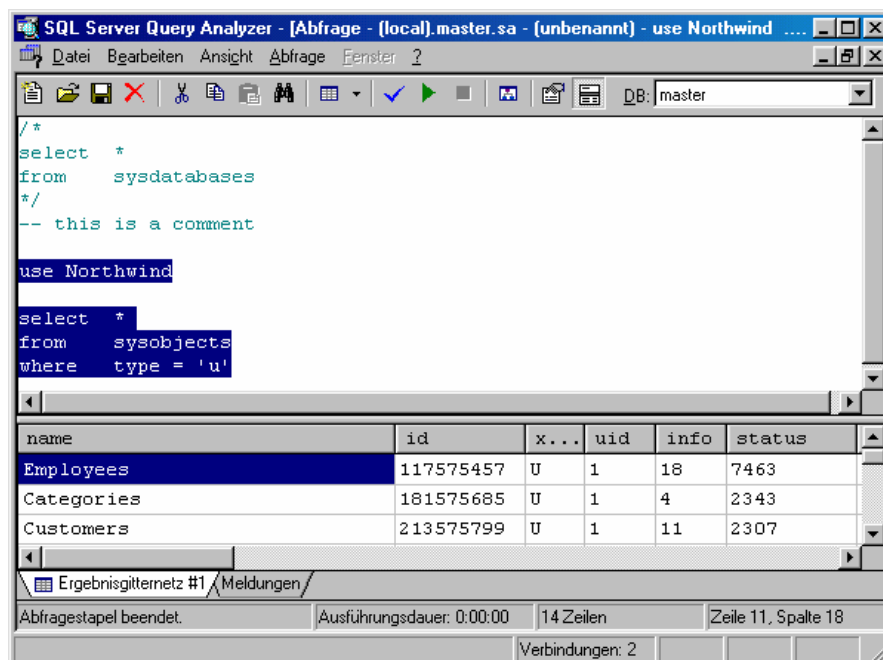
3. Geben Sie im Abfragefenster folgendes ein:
Select @@version
4. Klicken Sie auf der Symbolleiste auf **Abfrage ausführen** (grünes Dreieck), Shortcut = F5. Im Ergebnisbereich werden von der Abfrage Informationen über die verwendete Version von SQL Server zurückgegeben.
5. Wählen Sie im Listenfeld **DB** die Northwind-Datenbank aus. Geben Sie im Abfragefenster folgendes ein:
Select * from Customers
6. Klicken Sie auf der Symbolleiste auf **Abfrage ausführen**. Das Resultset erscheint im Ergebnisbereich. Die Ausgabe wird als Freiform-Text dargestellt.
7. Klicken Sie auf der Symbolleiste auf **Neue Abfrage**. Ein zweites Abfragefenster wird geöffnet, in dem eine separate Verbindung zu SQL Server hergestellt wird.

8. Wählen Sie im neuen Abfragefenster im Listenfeld DB die **Master**-Datenbank aus, wodurch sie zur aktuellen Datenbank für das zweite Abfragefenster wird. Northwind bleibt weiterhin die aktuelle Datenbank für das erste Abfragefenster.
9. Mit folgendem Statement kriegen Sie eine Liste aller Datenbanken auf Ihrem lokalen Server:
`Select * from Sysdatabases`
Dies ist ein weiteres Beispiel dafür, dass selbst die **Metadaten** in rel. Tabellen gespeichert sind.
10. - Statt im Listenfeld DB die richtige Auswahl zu treffen, kann mit **use Datenbankname** programmatisch erzwungen werden, dass die richtige Datenbank angesprochen wird.
Siehe Beispiel unten.

- Mit **Select * from sysobjects where type = "U"** können alle User-Tabellen aufgelistet werden.
Siehe Beispiel unten.

- Code kann man mehrzeilig ausklammern mit **/* statements */** oder einzeilig mit **2 vorangestellten Bindestrichen**.
Siehe Beispiel unten.

- Alternativ kann nur der **Code selektiert** (high lighted) werden, der auch ausgeführt werden soll.
Siehe Beispiel unten



3.7.b So erstellen Sie eine Datenbank mit SQL

1. Starten Sie SQL Server Query Analyzer.
2. Erstellen Sie eine neue Datenbank, mit Default – Einstellungen:
Create Database myDatabase
3. Überprüfen Sie die Eigenschaften Ihrer neuen Datenbank mit:
sp_helpDB myDatabase
Mit `Select * from Sysdatabases` in der Master-DB kriegen Sie eine Liste aller Datenbanken.
4. Möchten Sie andere als Default-Eigenschaften, können Sie dies explizit beim Erstellen der Datenbank angeben. Gucken Sie nach unter:
Start / Programme / Microsoft SQL Server / Onlinedokumentation
Unter Index / Create Database kriegen Sie alle Informationen aufgelistet.
6. Löschen Sie Ihre Datenbank wieder mit:
Drop Database sample_db

Vergewissern Sie sich mit **sp_helpDB**, dass Ihre Datenbank gelöscht wurde.

Erstellen Sie eine Datenbank mit dem Enterprise Manager

1. Öffnen Sie Start / Programme / Microsoft SQL Server / Enterprise Manager.
2. Öffnen Sie zuerst die Servergruppe und anschliessend Ihren lokalen Server.
3. Versuchen Sie selber herauszufinden, wie man ohne SQL eine Datenbank erstellen und auch wieder löschen kann.

3.7.c So benutzen Sie Constraints

Primary Key und CHECK Constraints

In dieser Tabellendefinition wird zusätzlich ein PRIMARY KEY– und ein CHECK – Constraint definiert. Die Check-Einschränkung stellt sicher, dass der Status nur Contract oder Employee sein darf.

```
use myOwnDB
Create Table authors
(
    id            int            not null,
    name          varchar(20)    not null,
    status        char(10)       not null,

    CONSTRAINT PK_id PRIMARY KEY (id),
    CONSTRAINT CHK_stat CHECK ( status in ('CONTRACT', 'EMPLOYEE'))
)
```

Aufgabe

Kreieren Sie eine neue DB "myOwnDB" und definieren Sie die Tabelle "authors" im Query Analyzer und fügen Sie anschliessend je einen gültigen Datensatz und einen, der gegen die Check-Einschränkung verstösst, ein. Ist der status case-sensitiv?

Default Constraint

Sie können Constraints erstellen, ändern und löschen, ohne eine Tabelle löschen und neu erstellen zu müssen.

Im folgenden Beispiel wird eine DEFAULT-Einschränkung hinzugefügt. Durch das Constraint wird der Wert 'Unknown' in die Spalte **name** eingegeben, wenn beim **Insert** kein Name mitgegeben wurde.

```
use myOwnDB

Alter table authors
Add
CONSTRAINT DEF_name DEFAULT 'Unknown' FOR name
```

Aufgabe

Versuchen Sie dieses Beispiel nachzuvollziehen und beweisen Sie, dass es funktioniert.

Foreign Key Constraint

Folgende Tabellendefinition hat eine Referenz auf sich selber: Manager ist Chef von Employee.

```
create table employee
(
    emp_no    int    not null,
    mgr_no    int    null,

    CONSTRAINT PK_emp PRIMARY KEY(emp_no),
    CONSTRAINT FK_mgr FOREIGN KEY(mgr_no) REFERENCES employee(emp_no)
)
```

Aufgabe

1. Erstellen Sie diese Tabelle und fügen Sie mindestens 3 Datensätze ein.
2. Versuchen Sie einen Datensatz zu löschen, auf den ein FK zeigt.
3. Entfernen Sie diesen Foreign Key Constraint (mit Alter Table, suchen Sie die Syntax in der Online Dokumentation) und löschen Sie dann einen Datensatz.

3.7.d So verwenden Sie die IDENTITY – Eigenschaft

Sie können die IDENTITY – Eigenschaft zum Erstellen von Spalten verwenden, die vom System generierte, sequentielle, numerische Werte enthalten, die sämtliche in eine Tabelle eingefügten Zeilen identifizieren. Eine solche Identitätsspalte wird oft für primäre Schlüsselwerte verwendet.

Wenn Schlüsselwerte automatisch von SQL Server und nicht von der Clientanwendung bereitgestellt werden, kann dies zu einer Leistungsverbesserung führen. Es vereinfacht die Programmierung und führt zu kürzeren primären Schlüsselwerten.

Syntax

IDENTITY(erster Wert, Inkrement)

Folgende Einschränkungen gelten:

- Pro Tabelle ist nur eine Identitätsspalte zulässig.
- Eine Identitätsspalte kann nicht aktualisiert werden (mit update).
- Eine Identitätsspalte kann keine NULL-Werte enthalten.
- Eine Identitätsspalte muss mit dem Datentyp integer verwendet werden.

Aufgabe

1. Führen Sie im Query Analyzer folgenden Statement aus (verwenden Sie eine eigene DB):

```
Create Table class
(
    student_id int          IDENTITY(1,1) NOT NULL,
    name       varchar(16)  NOT NULL,
    plz        int          NULL
)
```

2. Fügen Sie mit dem INSERT-Befehl mindestens 3 Zeilen ein. Was passiert, wenn Sie auch die student_id mitgeben wollen?
3. Gucken Sie sich die Tabelle mit Select ... an.

3.7.e So verwenden Sie die Funktion NEWID() und den Datentyp uniqueidentifier

Der Datentyp **uniqueidentifier** speichert eine global eindeutige Identifikationsnummer als 16-Byte-Wert (128 Bit).

Die Funktion **NEWID()** generiert einen solch eindeutigen Bezeichner (GUID), der unter Verwendung des Datentyps uniqueidentifier gespeichert werden kann. Derselbe GUID wird weltweit kein zweites Mal erstellt und kann daher zur eindeutigen Identifizierung von Zeilen in verschiedenen Tabellen, Datenbanken, Servern und Organisationen verwendet werden.

Folgende GUID ist ein Beispiel: 24F1B644-9DD7-11D2-993C-204C4F4F5020

Aufgabe

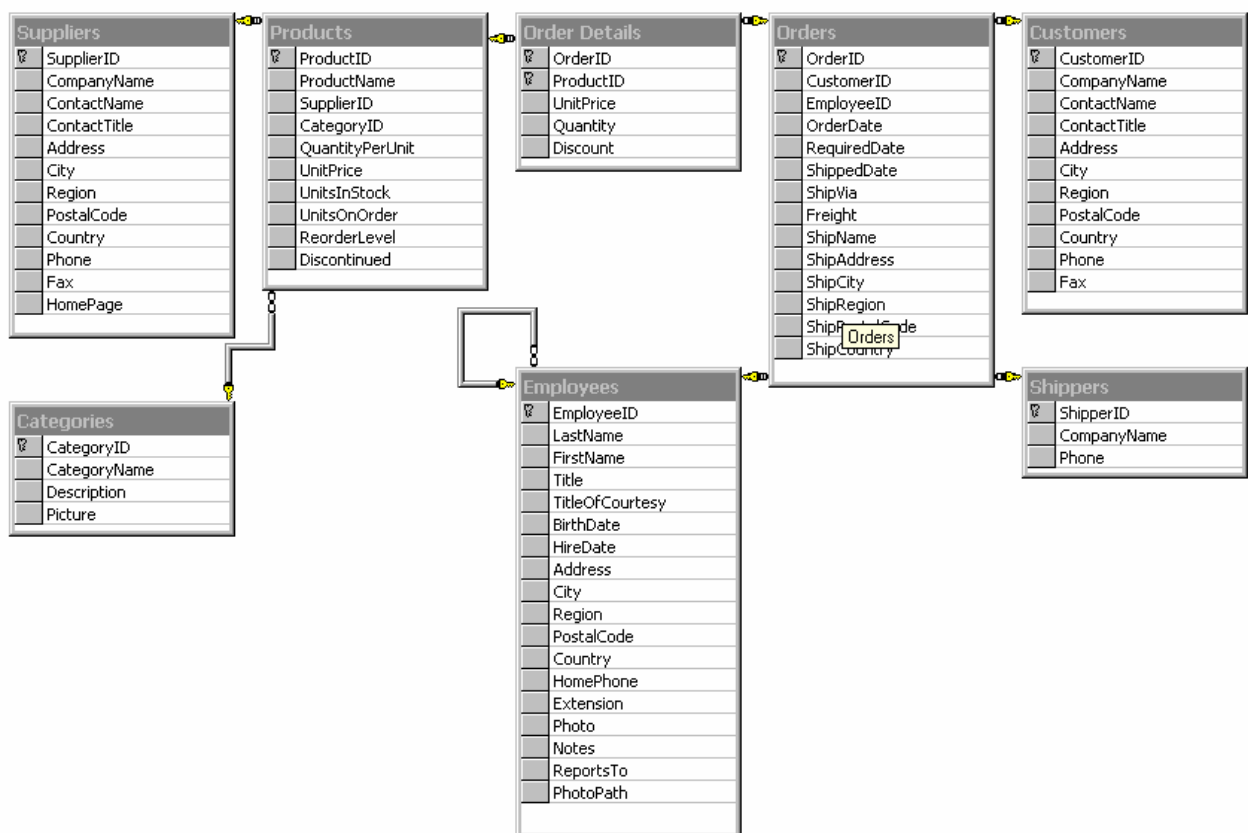
1. Führen Sie im Query Analyzer folgendes Statement aus:

```
create table customer
(
    Id          uniqueidentifier NOT NULL DEFAULT NEWID(),
    name        varchar(30)      NOT NULL,
    phone       varchar(10)      NULL
)
```

2. Fügen Sie mit dem INSERT-Befehl mindestens 3 Zeilen ein.
3. Gucken Sie sich die Tabelle mit Select ... an.

3.7.f So erstellen Sie ein ERM der Northwind – Datenbank

1. Öffnen Sie Start / Programme / Microsoft SQL Server / Enterprise Manager
2. Gehen Sie zu ..\ SQL Server-Gruppe \ lokalen Server \ Datenbanken \ Northwind \ Diagramme
3. Öffnen Sie mit der rechten Maustaste das Menü „Neues Datenbankdiagramm...“
4. Mit einem Assistenten werden Sie geführt. Wählen Sie alle Benutzer-Tabellen (alle Tabellen, die nicht mit Sys* beginnen).
5. Arrangieren Sie die Tabellen und deren Grösse so, dass alles gut sichtbar ist.
6. Speichern Sie dieses ERM, so dass Sie jederzeit wieder darauf zurückgreifen können.



3.7.g So erstellen Sie eine View

1. Öffnen Sie ein Abfragefenster in "SQL Query Analyzer" und selektieren Sie in der ComboBox DB die Datenbank **Northwind**.
2. Erstellen Sie ein Select Statement um alle Angestellte aufzulisten, die mit einem Einkauf mehr als 1000\$ ausgegeben haben. Diese Aufgabe nimmt Bezug auf das Northwind-ERM der vorhergehenden Seite.

Die Liste soll den Namen des Angestellten, den Produktnamen sowie die Ausgaben (UnitPrice * Quantity) enthalten.
Hinweis: die Tabelle [Order Details] muss in eckigen Klammern sein, da sie einen Leerschlag enthält.
3. Sind Sie zufrieden mit der Abfrage, machen Sie daraus eine View, indem Sie **Create View GoodEmployees As** vor das Select Statement hinzufügen.
5. Speichern Sie diese View in der Datenbank Northwind indem Sie sie ausführen.
6. Wenn Sie dieses Create View Statement nochmals ausführen, kriegen Sie eine Fehlermeldung, die da sagt, dass eine View mit diesem Namen schon auf der Datenbank existiert. Um eine bestehende View auch abändern zu können, wird üblicherweise zuerst mit Hilfe der Systemtabelle „sysobjects“ überprüft, ob diese View schon existiert und wenn ja, wird sie gelöscht. Fügen Sie folgendes Statement **vor** das Create View ... hinzu.

```
if exists (      select *
                from    sysobjects
                where   name = 'GoodEmployees'
                and     type = 'V' )
drop view GoodEmployees
go
```

7. Um sicherzustellen, dass mit der Northwind DB gearbeitet wird, fügen Sie zu oberst folgendes Statement hinzu.

```
USE Northwind
Go
```

8. Führen Sie diese View in einem neuen Abfragefenster aus mit:

```
Select * from GoodEmployees
```

9. Im **SQL Server Enterprise Manager** unter Server-Gruppe / lokaler Server / Datenbanken / Northwind / Sichten muss diese "GoodEmployees" nun sichtbar und mit einem Doppelklick auch editierbar sein. Nötigenfalls muss der Refresh Button aktiviert werden.

3.7.h So funktioniert eine Transaktion

In dieser Übung werden Sie mit den Auswirkungen eines Rollbacks und eines Commits vertraut gemacht.

1. Öffnen Sie den Query Analyzer.
2. In der Datenbank Northwind und der Tabelle Products sollen Sie den Preis von 'Tofu' um 10% erhöhen.
3. Dieser Update soll innerhalb einer Transaktion stattfinden.
4. Beweisen Sie, dass mit einem Rollback diese Transaktion rückgängig gemacht wird.
5. Beweisen Sie, dass mit einem Commit die Änderung permanent auf der Datenbank gespeichert ist.

Unten stehender Code dient als Vorlage.

```
use Northwind

Print 'Before Rollback'
Select Unitprice From Products Where ProductName = 'Chocolade'

Begin Tran
    Update Products
    Set    UnitPrice = UnitPrice * 1.25
    Where ProductName = 'Chocolade'
Rollback Tran

Print 'After Rollback'
Select Unitprice From Products Where ProductName = 'Chocolade'

Begin Tran
    Update Products
    Set    UnitPrice = UnitPrice * 1.25
    Where ProductName = 'Chocolade'
Commit Tran

Print 'After Commit'
Select Unitprice From Products Where ProductName = 'Chocolade'
```

Mit folgender Übung zeigen Sie, dass eine offene Transaktion andere Abfragen auf dieselben Daten sperrt, bis die offene Transaktion geschlossen wird.

1. Führen Sie folgenden Code im Query Analyzer durch. Das Commit fehlt hier bewusst, die Transaktion ist so noch offen.

```
Use Northwind

Set Transaction Isolation Level Serializable

Begin Tran
    Update Products
    Set    UnitPrice = UnitPrice / 1.1
    Where ProductName = 'Tofu'
```

2. Öffnen Sie ein neues Abfrage-Fenster unter dem Menü Abfrage.

3. In diesem neuen Fenster führen Sie folgende Abfrage aus.

```
use Northwind

Begin Tran
    Update Products
    Set    UnitPrice = UnitPrice * 1.1
    Where ProductName = 'Tofu'

    Select ProductName From Products
Commit Tran
```

4. Das Result Set lässt auf sich warten, die Abfrage ist hängig.

5. Im ersten Abfrage-Fenster lösen Sie nun das Commit aus und beenden damit die Transaktion.

```
Commit Tran
```

6. Das Result Set der hängigen Abfrage ist damit auch ausgelöst worden.

```
ProductName
-----
Alice Mutton
Aniseed Syrup
Boston Crab Meat
Camembert Pierrot
Carnarvon Tigers
Chai
Chang
```

Schliessen Sie bitte den Query Analyzer, damit allfällig offene Transaktionen die nachfolgenden Übungen nicht verfälschen.

3.7.i So erstellen Sie eine Stored Procedure

1. Schreiben Sie eine Stored Procedure mit dem Namen 'GetAllProducts' und dem Aufrufparameter 'product'.
2. Diese STP soll neben dem gewünschten Produkt noch sämtliche andere Produkte des jeweiligen Suppliers auflisten.
Beispiel: **GetAllProducts 'Tofu'** liefert das gesamte Produktsortiment des Lieferanten von Tofu.
3. Fangen Sie ungültige Parameter ab.
4. Die Abfrage soll mit einer lokalen Variable und nicht mit einer Unterabfrage gelöst werden.
5. Rufen Sie die STP mit verschiedenen Parametern auf. Provozieren Sie auch einen Fehler mit einem leeren String als Parameter.
GetAllProducts 'Chocolade' oder **GetAllProducts ''**
6. Wechseln Sie zum SQL Server Enterprise Manager unter Programme / Microsoft SQL Server. Unter lokalen Server / Datenbanken / Northwind / Gespeicherete Procedures sollten Sie GetAllProducts aufgelistet haben.

Untenstehender Code dient als Vorlage:

```
-- stelle sicher, dass die Northwind-DB benutzt wird
use northwind
go

-- wenn die STP schon existiert, lösche sie zuerst
if exists (select * from sysobjects where name = 'GetAllProducts' and type = 'P')
    drop Procedure GetAllProducts
go

-- definiere STP
Create Procedure GetAllProducts (@product varchar(20))
AS

-- error handling
if ( @product = '' ) /* leerer string */
begin
    select 'Sie müssen einen Produktnamen übergeben' as error
end

else
begin
    -- deklariere lokale Variable
    declare @supplierID int

    -- speichere id des Suppliers, der das gewünschte Produkt vertritt
    select @supplierID = s.SupplierID
    from suppliers s, Products p
    where p.SupplierID = s.supplierID
    and p.productname = @product

    -- liste alle produkte dieses Suppliers
    select ProductName from products where supplierID = @supplierID
end
```

3.7.j So erstellen Sie einen Trigger

In dieser Übung erstellen Sie einen Trigger, der eine Warnung auslöst, wenn der Bestand eines Artikels im Lager nach Abzug der offenen Bestellungen unter eine kritische Grösse fällt.

```
use northwind
go

-- lösche trigger, wenn er schon existiert
if exists ( select *
            from   sysobjects
            where  type = 'TR'
            and    name = 'CheckStock')
    drop trigger CheckStock
go

-- definiere trigger
Create Trigger CheckStock
On products For Update
As
if exists ( select *
            from   inserted
            where  UnitsOnOrder > (UnitsInStock + 10) )
begin
    -- warning
    declare @product varchar(30)
    select  @product = productName from inserted where UnitsOnOrder > (UnitsInStock + 10)
    print   'Achtung: Anzahl '+' @product +' im Lager ist unter kritische Grösse gefallen'
end
```

So testen Sie den Trigger

Bestellen Sie mit je einem Update auf das Feld UnitsOnOrder 100 Stück 'Tofu' und 80 Stück 'Chocolade', so dass der Trigger ausgelöst wird.

So holen Sie Informationen über den Trigger

Mit folgenden Systemprozeduren können Sie alle Informationen über einen Trigger abfragen.

- sp_helptext CheckStock -- Zeigt den Code des Triggers
- sp_helptrigger Products -- Zeigt alle Trigger dieser Tabelle