3 Grundlagen Datenbank

3.1 Datenbanken

3.1.1 Einführung in die Datenbanktheorie

Das Folgende Kapitel wendet sich speziell an den Personenkreis der in die Theorie von Datenbanken noch nicht so eingedrungen ist, beziehungsweise dient als Nachschlagwerk für die verschiedenen Begriffe. Man kann also bei entsprechenden Vorwissen die folgenden erklärenden Kapitel überspringen und bei Bedarf nachschlagen.

Begriffe

Um Missverständnisse auszuschließen und zu einer gemeinsamen Sprachregelung zu kommen, sollte man die verwendeten Begriffe möglichst genau definieren:

- Eine Datenmenge ist eine Menge von einzelnen Datensätzen. Jeder Datensatz besteht aus mindesten einem Feld. Die Herkunft dieser Datenmenge ist nicht festgelegt.
- Eine Tabelle ist als Datenbankbestandteil eine spezielle Ausführung einer Datenmenge. Die Tabelle speichert als physisches vorhandenes Element die Daten.
- Eine Abfrage ist eine virtuelle Datenmenge, die den Inhalt von tatsächlich vorhandenen Tabellen in einer frei wählbaren Anordnung abbildet, manchmal auch Projektion genannt.
- Eine Datenbank ist eine Zusammenstellung von logisch zusammengehörigen Tabellen.
- Ein Datenbankserver verwaltet verschiedene Datenbanken und stellt auch das Management, die Sicherung und andere Verwaltungsdienste zur Verfügung.

Wichtige Informationen bei der Entwicklung einer Datenbankanwendung sind das Verhalten der Datenbank bzw. des Datenbankservers selbst.

Was ist eine Datenbank

Eine Datenbank ist eine geordnete Sammlung von Daten, die auf irgendeine Weise miteinander in Beziehung stehen.

Die ersten Generationen von Datenbanken waren sogenannte File-Systeme. Zuerst auf Band, dann auch auf Festplatten. In diesem Datei-Systemen wurden die Daten nacheinander abgespeichert. Um auf einen bestimmten Datensatz zu zugreifen, muss man an den Anfang der Datei (oder Bandes) gehen und anschließend alle Datensätze durchlaufen, bis man den richtigen gefunden hat. Somit ist auch klar, das ein einfaches sortieren oder einfügen von Daten in eine

sortierte Datenmenge enormen Aufwand und Kapazität erfordert hat. Um diesen Beschränkungen zu entfliehen, (und durch neue, schnellere und größere Festplatten ermöglicht) haben sich aus diesen System die heutigen relationalen oder objektorientierten Datenbanken entwickelt. Die derzeit am meisten verwendeten Datenbanken sind die relationalen und im weiteren werde ich nur noch diese behandeln.

Desktop und Client-Server Datenbankarten

Gerade der Begriff Netzwerkdatenbank ist sehr verwirrend. Wird damit eine Access-Datenbank, die mehrere Benutzer über das Netzwerk verwenden so bezeichnet ? Oder eine Serverbasierende Datenbank? Die folgenden Erklärungen sollten die Begriffe klarer werden lassen.

Stand-Alone Datenbank Eine Stand-Alone Datenbank ist eine Desktop-Datenbank, es befinden sich daher die Daten auf dem Arbeitsplatzrechner. Auf die Daten kann immer nur ein Anwender, mit immer nur einem Programm zugreifen. Es ist zwar prinzipiell möglich über ein Netzwerk auf die Datenbankdatei zuzugreifen, aber es kann der eine nur in der Datenbank arbeiten, wenn der andere sein Programm geschlossen hat. Probleme die durch den gleichzeitigen Zugriff entstehen können daher gar nicht auftreten. Bei jeder etwas umfangreicheren Datenbank wird dieses Verhalten zu einem Engpass. Man stelle sich nur vor, der eine Benutzer öffnet die Datenbankdatei und vergisst auf das schließen des Programms. Kein anderer Benutzer kann die Daten in der Zwischenzeit benutzen!

File-Share Datenbank Moderne Netzwerke bieten die Möglichkeit, dass mehrer Anwender auf ein und dieselbe Datei zugreifen. Auf diese Weise ist es auch möglich das mehrer Programme auf ein und dieselbe Datenbankdatei zugreifen. Diese Version der Desktop-Datenbank nennt man File-Share Datenbank und damit ist bereits ein echter Mehrbenutzer Betrieb möglich. Das ganze hat jedoch (unter anderem) einen entscheidenden Nachteil: Die Datenverarbeitung erfolgt auf den Arbeitsplatzrechnern. Für Abfragen muss der jeweilige ganze Datenbestand zu den Arbeitsplatzrechnern gebracht werden, dementsprechend hoch ist die Belastung für das Netzwerk. Weiter können auch Störungen am Netzwerk leicht zu Datenverlust bzw. zu inkonsistenz der Datenbeständen führen.

Client-Server Datenbank Bei Client-Server Datenbanken hat nur der Datenbankserver selbst direkten Zugriff auf die Dateien des Datenbestandes. Anfragen werden somit direkt an den Datenbankserver gestellt, von diesem bearbeitet und die Ergebnisse an den Arbeitsplatzrechner zurückgeliefert. Die Verwaltung beim gleichzeitigen Zugriff durch mehrer Arbeitsplatzrechner obliegt dem Datenbankserver. Falls eine Verbindung durch eine Störung abbricht, so wird dieses erkannt und die noch nicht kompletten Anfragen verworfen und somit die Konsistenz der Daten erhalten. Gerade zur File-Share Datenbank können Netzbelastungen drastisch gesenkt werden. Man stelle sich nur vor, das man den größten Wert aus einer unsortierten Tabelle mit 1 Millionen Datensätze habe will. Bei der File-Share Datenbank müssen alle Datensätze geholt und bearbeitet werden, bei der Client Server Datenbank nur das Ergebnis. Weitere Bereiche sind die Möglichkeit Backups zu erstellen während die Datenbank weiter in Verwendung ist.

Relationale Datenbanken

Der Begriff relationale Datenbank geht auf einen Artikel von E. F. Codd zurück, der 1970 veröffentlich wurde. Codd bezeichnet Datenbanken als "minimal relational", wenn sie folgende Bedingungen erfüllen. [list] [*]Die Informationen werden einheitlich in Form von Tabellen repräsentiert. [*]Der Anwender sieht keine Verweisstrukturen zwischen den Tabellen. [*]Es gibt mindestens die Operation der Selektion, der Projektion und des JOIN definiert. [/list] 1985 veröffentlichte Codd zwölf Regeln, die relationalen Datenbanken im strengeren Sinn definieren, Ende 1990 veröffentlichte er ein Buch über relationale Datenbanken, in dem er die einstigen zwölf Regeln des relationalen Modells auf 333 Regeln differenziert. Dadurch wird die Relationalität einer Datenbank bis ins letzte Detail festgeschrieben Soweit die Theorie, normalerweise sprechen Hersteller von relationalen Datenbanken, wenn die Mehrheit der 12 Regeln eingehalten werden.

Begriffe in relationalen Datenbanken Man kann nicht über relationale Datenbanken sprechen, ohne zuvor einige Begriffe zu klären.

Relationen Eine Relation ist gleich einer Tabelle. Die Daten werden daher in Relationen gespeichert.

Attribut und Tuples Die Attribute sind die Spalten einer Relation (Tabelle), die Tuples sind die Datensätze.

Degree und Kardinalität Die Zahl der Attribute einer Relation nennt man Degree, das ist der Ausdehnungsgrad. Die Zahl der Tuples ist die Kardinalität. Eine Relation mit Degree Null macht keinen Sinn, eine Kardinalität von NULL Tuples hingegen ist eine leere Relation.

Domain Eine Domain ist ein Wertebereich. Man kann z.B. eine Domain Nachnamen vom Type „Zeichen mit Länge 20“ erstellen. Diese wird immer dann verwendet wenn man Datenspalten mit dem Type „Nachname“ erstellen muss. Warum dieser Umweg? Wenn man später Tabellen miteinander verknüpft (in Relation bringt), so müssen die Spalten (Attribute) der gleichen Domain unterliegen. Habe ich vorher eine Domain definiert, so gibt es keine Probleme. Weiter ist es kein Problem wenn man draufkommt das die „Nachnamen“ länger sind, so kann die Definition der Domain geändert werden und alle Spalten haben die richtige Länge. Würde ich beim händischen Editieren eine Spalte vergessen so würde die Datenbank nicht mehr korrekt arbeiten.

NULL Ein „Nichtwert“ der anfangs immer für Verwirrung sorgt ist NULL. NULL ist nicht gleich Null! Ein besserer Ausdruck wäre UNBEKANNT. NULL macht übrigens aus einer zweiwertigen Logik (Ja/Nein) eine dreiwertige (Ja/Nein/Unbekannt). Vorstellen kann man es sich am besten mit einem Beispiel. Jedem Konferenzraum kann ein Beamer mit seiner Nummer zugeteilt werden. Die Räume welche keinen Beamer besitzen (zuwenige Beamer vorhanden) bekommen als Wert NULL zugeteilt, da ja ein nicht vorhandener Beamer auch keine Nummer besitzt, folglich also unbekannt ist.

Schlüssel Im Zuge der Normalisierung (welche später erklärt wird) werden die Daten auf viele Tabellen vereilt. Um dieses Tabellen wieder richtig in Relation zu bringen werden verschiedene Schlüssel, auch Keys genannt, verwendet.

Primärschlüssel (Primary Key) Jede Relation (Tabelle) besitzt einen Primärschlüssel um einen Datensatz eindeutig zu identifizieren. Ausnahmen von dieser Regel gibt es nur bei „M:N Verknüpfungen“ für die Zwischentabellen verwendet werden (die meisten Datenbanksysteme können diese Verknüpfungen nicht direkt abbilden). Ein Primärschlüssel ist immer eindeutig und ohne Duplikate. Meistens wird dafür eine fortlaufende Nummer verwendet, ist aber nicht zwingend. Vorsicht bei bestimmten Fällen, denn selbst Sozialversicherungsnummern müssen nicht eindeutig sein !

Sekundärschlüssel (Secundary Keys) Werden dafür verwendet um bei bestimmten Datenbankoperationen die Effizienz zu steigern, da die Datensätze intern nicht ungeordnet sondern in sortierter Reihenfolge verwaltet werden. Beim verwenden sollte aber immer auf die zugrunde liegende Datenbank Rücksicht genommen werden, da die Vor- und Nachteile stark datenbankabhängig sind.

Fremdschlüssel (Foreign Key) Ist der Verweis in der Tabelle auf einen Primärschlüssel in einer anderen Tabelle. Gerade in relationalen Datenbanken gibt es sehr viele Verknüpfungen die auf Primär- und Fremdschlüsselpaaren aufbauen. Wichtig ist, das Primär- und Fremdschlüssel der gleichen Domain unterliegen.

Referentielle Integrität Die referentielle Integrität stellt sicher das die Daten zueinander (über Primär- und Fremdschlüssel) glaubhaft bleiben. Ein einfügen, ändern oder löschen ist zu verweigern wenn dadurch die Datenintegrität verletzt würde. Man kann zum Beispiel keine Datensätze aus der Personentabelle löschen, solange in der Tabelle der Bestelllungen auf diese Personen verwiesen wird.

Normalisierung Unter der Normalisierung einer Datenbank, wird die Technik des logischen Datenbankdesigns bezeichnet. Es erfolgt meistens durch das Schrittweise optimieren der Datenbank zur Designzeit. Theoretiker haben insgesamt 5 Stufen der Normalisierung herausgearbeitet, wobei in der Praxis meist nur die ersten 3 Stufen verwendet werden.

Warum wird meistens nur bis zur 3. Normalform normalisiert? Bei der Anwendungsentwicklung besteht meistens nicht das Ziel, möglichst den exakten theoretische Grundlagen zu entsprechen, sondern eine möglichst effiziente Komplettlösung für das Problem zu erhalten. Dazu gehört natürlich auch, die Rücksicht auf das verwendete Datenbanksystem und die damit zu erzielenden Performance. Es leuchtet jedem ein, das die Aufteilung der Informationen auf viele Tabellen bei einer Auswertung zu schlechteren Ergebnissen führt. Somit kann es sein, das in Teilbereichen sogar eine Denormalisierung aus Performancegründen nötig ist, oder der gewonnene Platz bzw. das Geschäftsmodell keine Normalisierung sinnvoll erscheinen lassen.

Ausgangslage	Alle Informationen in einer Tabelle
1. Normalform	Jede Spalte einer Tabelle enthält unteilbare Informationen. Die Datensätze verwenden keine sich wiederholenden Informationen, die nicht auch zu einer separaten Gruppe zusammengefasst werden könnten
2 Normalform	Es wird die 1. Normalform eingehalten und alle Informationen in nicht Schlüsselfeldern hängen nur vom kompletten Primärschlüssel ab
3 Normalform	Es wird die 2 Normalform eingehalten und alle Informationen in den nicht Schlüsselfeldern sind untereinander nicht abhängig.
4 Normalform	Es wird die 3. Normalform eingehalten und in gleichen Tabellen sind keine unabhängigen Objekte vorhanden, zwischen denen eine m:n Beziehung bestehen könnte
5 Normalform	Die normalisierte Datenbank kann nicht weiter in Tabellen mit weniger Attributen konvertiert werden. Es muss sich jederzeit der unnormalisierte Ursprungszustand ohne Informationsverlust herstellen lassen

Tabelle 3.1: Datenbank Normalisierungstufen Übersicht

Grunddaten

Mit Grunddaten werden die Informationen bezeichnet, die Voraussetzung für die tägliche Arbeit sind und während des täglichen Betriebs anfallen. Sie stellen die Basis des Datenbanksystems dar. Die Grunddaten werden in zwei Kategorien, Stammdaten und Bewegungsdaten, eingeteilt.

Stammdaten Stammdaten sind diejenigen Grunddaten, die über einen längeren Zeitraum benötigt werden. Sie bilden den Grundbestand an Daten für das Datenbanksystem und werden auch als Bestandsdaten bezeichnet. Stammdaten weisen eine geringe Änderungshäufigkeit auf. Üblicherweise ist bei der Neuanlage von Stammdaten noch nicht bekannt, wann Änderungen zu erwarten und wie lange die Daten insgesamt gültig sind. Da auf Stammdaten häufig zugegriffen wird, ist ihre aktuelle Pflege notwendig, so dass Änderungen unmittelbar im Datenbestand nachgezogen werden sollten. Somit ist auch die normale Zugriffsart festgelegt, auf Stammdaten wir meistens in Verbindung mit Abfragen lesend zugegriffen, nur die Wartung erfolgt schreibend.

Bewegungsdaten Im Gegensatz zu Stammdaten haben Bewegungsdaten eine begrenzte Lebensdauer, die durch einen vorgegebenen Lebenszyklus beschrieben ist. Bewegungsdaten haben einen konkreten Zeitbezug, der für die Bedeutung und Interpretation der Information wichtig ist. Des weiteren beziehen sich Bewegungsdaten auf Stammdaten, weshalb sie auch als abgeleitete Daten bezeichnet werden. Da Bewegungsdaten gegenüber Stammdaten in der Menge mächtiger sind, ist gerade hier auf ein gutes Design zu achten. Betrachten wir es am Beispiel eines Zählers einer Station: Die Zähler werden im 10 Minutenrhythmus in die Bewegungsdaten eingefügt, das sind 144 Datensätze pro Tag, währenddessen der Stammdatenteil gleich bleibt, nämlich 1

Datensatz.

```
Version: $LastChangedRevision: 55 $
```
[1]

[1] Autor: Andreas Frieß
Lizenz: GFDL

3.1.2 DDL Datendefinitionssprache

Die Datendefinitionssprache (Data Definition Language = DDL[2]) umfasst die Sprachteile von Datenbanksprache, mit deren Hilfe man Datenstrukturen wie Tabellen und andere ähnliche Elemente erzeugt. Es sind das die Befehle die mit *CREATE* beginnen, zum Beispiel *CREATE TABLE* und *CREATE INDEX*.

3.1.3 DML Datenveränderungssprache

Die Datenveränderungssprache (Data Manipulation Language = DML[3]) umfasst die Sprachteile von Datenbanksprache, die sich mit dem Lesen, Ändern und Löschen von Daten beschäftigen. Es sind das die Befehle *SELECT*, *INSERT*, *UPDATE* und *DELETE*.

Im folgend sehen wir uns die wichtigsten Befehle an, wobei ein Befehl besonders mächtig ist—*SELECT*—.

SELECT

SELECT ist einer der Vielseitigsten und am meisten eingesetzten Befehle überhaupt. Er dient zum Abfragen von Datenmengen aus der Datenbank. Er ermöglicht die Abfrage von Daten aus den Tabellen (die Projektion)

Einfachste Darstellung Bei der einfachsten Abfrage haben wir es mit wenigen Schlüsselwörter zu tun.

```
SELECT [DISTINCT] 'Auswahlliste'
  FROM 'Quelle'
  WHERE 'Where-Klausel'
  [GROUP BY ('Group-by-Attribut')+
  [HAVING 'Having-Klausel']]
  [ORDER BY (''Sortierungsattribut'' [ASC|DESC])+];
```

Mit SELECT wird die Abfrage eingeleitet, anschliessend kommt die Auswahlliste der gewünschten Tabellenspalten. Dann das FROM , das angibt welche Tabellen die Daten bereitstellen und das WHERE, das angibt nach welchen Kriterien die Daten vorselektiert werden.

SELECT 'Auswahlliste' Hier gibt man die einzelnen Tabellenspalten an, die sich später in der rückgelieferten Datenmenge finden. Sind die Name nicht eindeutig genug, besonders wenn mehrere Tabellen betroffen sind, so muß man den Tabellennamen und eventuell auch die Datenbank angeben, sprich die Eindeutigkeit qualifizieren. Somit kann ein SELECT so aussehen: `SELECT␣MyDB.STPerson.Vorname`, meistens wird die verkürzte Schreibweise verwendet wie `SELECT␣Vorname`.

In komplexeren Abfragen kann auch statt dem Tabellnamen alleine, eine Funktion stehen. Damit kann man in Datenmengen mathematische und statistische Funktionen verwenden. Auch

[2] siehe auch http://de.wikipedia.org/wiki/Data_Definition_Language
[3] siehe auch http://de.wikipedia.org/wiki/Data_Manipulation_Language

Verzweigungen und Berechnungen sind möglich. Zu diesen Punkten kommen dann an späterer Stelle die entsprechden Erklärungen.

DISTINCT erzwingt die Vermeidung von doppelten Datensätzen. Es überspringt in der Ausgabe die mehrfach vorkommenden Datensätze. Die Folge ist ein höherer Aufwand am Server, da das urspüngliche Ergebnis (ohne DISTINCT) meist noch entsprechend nachbearbeitet werden muß. Bevor man DISTINCT zur Korrektur unerklärlicher doppelter Datensätze verwendet (Verschleiern von Problemen), sollte man sich zuerst seine JOINS und WHERE Klauseln ganz genau ansehen, denn dort liegt meistens das Problem.

Eine gerne verwendete Abkürzung stellt das beliebte `SELECT␣*` dar. Hier darf dann das Programm zur Laufzeit erraten, welche Spalten es gibt und von welchen Typ sie sind. Es einfach zu verwenden, birgt aber in fertigen Programmen einiges an versteckten Fehlerquellen. Wird die dem SELECT zugrunde liegende Tabelle geändert oder auch nur Spalten getauscht, so kann es sein, daß Fehler nicht beim ausführen des Statements auffallen (Spalte xx nicht gefunden) und erst andere Programmteile dann unklare Fehlermeldung produzieren. Ich empfehle deshalb, die Spaltennamen wirklich anzugeben. Es gibt Programme die gerade diese Eigenschaft ausnutzen, dort wird es aber bewusst gemacht und entsprechend abgesichert.

FROM 'Quelle' FROM gibt die Quelle der Daten an. Es ist einfach der Tabellenname, eine Verbindung von Tabellen (JOIN) oder auch eine untergelagerte SELECT Anweisung.

WHERE 'Where-Klausel' Die WHERE - Klausel schränkt die Ergebnisdatenmenge ein. Bei einfachen Beispielen wird sie gerne, oft zu Unrecht, weggelassen. Wir müssen aber immer Datenmengen betrachten, das Ergebnis kann eine Zeile oder eine million Zeilen sein. Wenn wir in einer Adressdatenbank eine Personen suchen, so wird es trotzdem sinnvoll sein, von Haus aus die Suche einzuschränken. Denn alles was nicht an Daten sinnlos übertragen werden muß, spart Bandbreite, Speicher und erhöht die Geschwindigkeit.

GROUP BY ('Group-by-Attribut' Die Daten werden nach dem Group-by-Attributen zusammengefasst (gruppiert). Dazu müssen auch in der Auswahlliste, für alle nicht durch GROUP BY erfassten Spalten, entsprechende Operationen verwendet werden (SUM, AVG, MIN, ...).

HAVING 'Having-Klausel' Ist als ein WHERE zu betrachten, das allerdings erst **nach** dem Gruppieren wirksam ist und deshalb nur auf die Gruppierungsfunktionen wirksam ist.

ORDER BY ("Sortierungsattribut" [ASC|DESC]) Die ausgegeben Daten werden entsprechend sortiert. Das Sortierungsattribut sind die entsprechden Spaltennamen in der reihenfolge wie sortiert werden soll.

Beispiele zu SELECT

*SELECT * FROM st_person;*
Ohne Einschränkung oder Sortierung.

```
"STPerson","cVName","cFName","cMName","cRName"
378,"Hope","Giordano","HoGi","Hope0Giordano"
379,"Rose","Bruno","RoBr","Rose4Bruno"
380,"Lauren","Morgan","LaMo","Lauren4Morgan"
381,"Megan","Coleman","MeCo","Megan9Coleman"
```

*SELECT * FROM st_person where STPerson = 875;*
Die Person deren ID 875 ist.

```
"STPerson","cVName","cFName","cMName","cRName"
875,"Hannah","Collins","HaCo","Hannah3Collins"
```

*SELECT * FROM st_person where cVName like 'H%';*
Alle Personen deren Vorname mit „H“ beginnt. Das Prozentzeichen „%“ ist eine Wildcard. So wie bei manchen Betriebssystemen der Stern „*“.

```
"STPerson","cVName","cFName","cMName","cRName"
875,"Hannah","Collins","HaCo","Hannah3Collins"
406,"Hannah","Cook","HaCo","Hannah9Cook"
845,"Hannah","Doyle","HaDo","Hannah9Doyle"
1363,"Hannah","Foster","HaFo","Hannah6Foster"
```

*SELECT * FROM st_person order by cFName;*
Alle Personen, aber nach Familienname aufsteigend sortiert.

```
"STPerson","cVName","cFName","cMName","cRName"
1258,"Caitlin","Adams","CaAd","Caitlin4Adams"
687,"Taylor","Alexander","TaAl","Taylor5Alexander"
644,"Renee","Alexander","ReAl","Renee8Alexander"
885,"Taylor","Alexander","TaAl","Taylor3Alexander"
1131,"Sarah","Alexander","SaAl","Sarah5Alexander"
603,"Megan","Allen","MeAl","Megan0Allen"
1172,"Isabella","Allen","IsAl","Isabella0Allen"
472,"Isabelle","Allen","IsAl","Isabelle6Allen"
```

*SELECT * FROM st_person order by cFName desc;*
Alle Personen, aber nach Familienname absteigend sortiert.

```
"STPerson","cVName","cFName","cMName","cRName"
842,"Isabella","Young","IsYo","Isabella6Young"
1232,"Taylor","Young","TaYo","Taylor3Young"
427,"Ashley","Young","AsYo","Ashley3Young"
420,"Kyla","Young","KyYo","Kyla2Young"
668,"Alexandra","Wright","AlWr","Alexandra6Wright"
1070,"Madison","Wright","MaWr","Madison5Wright"
```

*SELECT * FROM st_person where cVName like 'H%' order by cFName;*
Alle Personen deren Vorname mit „H“ beginnt und nach Familienname aufsteigend sortiert.

```
"STPerson","cVName","cFName","cMName","cRName"
932,"Hope","Bell","HoBe","Hope3Bell"
1194,"Harmony","Campbell","HaCa","Harmony7Campbell"
515,"Harmony","Clark","HaCl","Harmony4Clark"
1279,"Holly","Collins","HoCo","Holly7Collins"
875,"Hannah","Collins","HaCo","Hannah3Collins"
406,"Hannah","Cook","HaCo","Hannah9Cook"
1296,"Harmony","Diaz","HaDi","Harmony1Diaz"
```

SELECT cVName, cFName FROM st_person where cVName like 'H%' order by cFName; Anzeige nur der Spalten „cVName“ und „cFName“ und alle Personen deren Vorname mit „H“ beginnt und nach Familienname aufsteigend sortiert.

```
"cVName","cFName"
"Hope","Bell"
"Harmony","Campbell"
"Harmony","Clark"
"Holly","Collins"
"Hannah","Collins"
"Hannah","Cook"
"Harmony","Diaz"
```

SELECT cVName, cFName FROM st_person where (cVName like 'H%') or (cVName like 'A%') order by cFName;
Anzeige nur der Spalten „cVName“ und „cFName“ und alle Personen deren Vorname mit „H“ oder „A“ beginnt und nach Familienname aufsteigend sortiert.

```
"Alyssa","Butler"
"Abby","Butler"
"Ashley","Butler"
"Ashley","Byrne"
"Harmony","Campbell"
"Harmony","Clark"
"Anna","Clark"
"Abby","Coleman"
"Hannah","Collins"
"Amelia","Collins"
"Anna","Collins"
```

SELECT count(cVName), cVName FROM st_person group by cVName order by count(cVName) desc; Anzahl der Vorkommen der Vornamen absteigend sortiert.

```
"count(cVName)","cVName"
19,"Amelia"
19,"Angel"
17,"Laura"
```

```
16,"Elizabeth"
16,"Paris"
16,"Ruby"
15,"Caitlin"
14,"Sophie"
14,"Abby"
14,"Mikayla"
```

SELECT count(cVName) as Anzahl, cVName as Vorname FROM st_person group by cVName order by count(cVName) desc; Anzahl der Vorkommen der Vornamen absteigend sortiert. Dazu noch die Spaltennamen geändert.

```
"Anzahl","Vorname"
19,"Amelia"
19,"Angel"
17,"Laura"
16,"Elizabeth"
16,"Paris"
16,"Ruby"
15,"Caitlin"
14,"Sophie"
14,"Abby"
14,"Mikayla"
```

SELECT count(cVName) as Anzahl, cVName as Vorname FROM st_person group by cVName desc having Anzahl = 16; Anzahl der Vorkommen der Vornamen, Dazu noch die Spaltennamen geändert und die Anzahl muß sechzehn sein

```
"Anzahl","Vorname"
16,"Elizabeth"
16,"Paris"
16,"Ruby"
```

INSERT

INSERT wird für das Einfügen von Datensätzen in eine Tabelle verwendet.

Einfache Darstellung Beim einfachen Einfügen haben wir es mit wenigen Schlüsselwörter zu tun.

```
INSERT 'Ziel' ('Spaltenliste')
  VALUES ('Werteliste')
```

Bei dem Ziel handelt es sich um die Tabelle in die die Daten eingefügt werden sollen. Die Spaltenliste kann auch weggelassen werden, wenn die Werteliste in der exakt gleichen Reihenfolge ist, wie die Definition in der Tabelle.

Wenn die Spaltenliste vorhanden ist, so müssen die Werte in der Werteliste in der gleichen Reihenfolge stehen. Es muß dann aber nicht die Reihenfolge und Anzahl der Spalten mit der Tabellen Definition übereinstimmen. Das wird normalerweise verwendet, da an Spalten mit automatischen Zählern (meist Primärschlüssel) keine Werte übergeben werden dürfen!

Beispiele zu INSERT

INSERT st_person (cVName,cFName,cMName, cRName) VALUES ("Hope","Giordano","HoGi", "Hope0Giordano"); Bei *INSERT* und anderen Befehlen zur Erstellung von Daten, gibt es kein Ergebnismenge.

```
SELECT * FROM st_person;
```

```
"STPerson","cVName","cFName","cMName","cRName"
1,"Hope","Giordano","HoGi","Hope0Giordano"
2,"Harmony","Campbell","HaCa","Harmony7Campbell"
3,"Harmony","Clark","HaCl","Harmony4Clark"
4,"Holly","Collins","HoCo","Holly7Collins"
```

Die Spalte STPerson, ist zwar beim INSERT nicht angegeben, wird vom System automatisch vergeben.

Ein weiteres Beispiel befindet unter Projekt MySQLTestData

UPDATE

Mit *UPDATE* können die Daten in den Tabellen geändert werden.

Einfache Darstellung Beim einfachen Ändern haben wir es mit wenigen Schlüsselwörter zu tun.

```
UPDATE 'Ziel' SET 'Spalte1' = 'Wert1' , 'Spalte2' = 'Wert2'
  WHERE 'Where-Klausel'
```

Bei dem Ziel handelt es sich um die Tabelle in der die Daten geändert werden sollen. Nach dem Schlüsselwort *SET* erfolgt die Auflistung der Spalten mit den neuen Wert.

Wichtig ist hier die *WHERE*-Klausel! Fehlt sie so werden **alle Datensätze** der Tabelle geändert! Soll nur ein einziges Datensatz von der Änderung geändert werden, so muß die Einschränkung richtig definiert sein. Normalerweise wird hier der Primärschlüssel (auch zusammengesetzt) verwendet, denn so ist die Eindeutigkeit sichergestellt. Bezüglich der Möglichkeiten der *WHERE*-Klausel bitte beim Befehl *SELECT* nachzulesen.

Beispiele zu UPDATE

UPDATE st_person set cRName = "HoGi"WHERE STPerson = 1; Bei *UPDATE* und anderen Befehlen zur Erstellung von Daten, gibt es kein Ergebnismenge.

```
SELECT * FROM st_person;

"STPerson","cVName","cFName","cMName","cRName"
1,"Hope","Giordano","HoGi","HoGi"
2,"Harmony","Campbell","HaCa","Harmony7Campbell"
3,"Harmony","Clark","HaCl","Harmony4Clark"
4,"Holly","Collins","HoCo","Holly7Collins"
```

UPDATE st_person set cVName = "Hopper", cRName = "HoGi1"WHERE STPerson = 1; Hier werden zwei Spalten gleichzeitig geändert

```
SELECT * FROM st_person;

"STPerson","cVName","cFName","cMName","cRName"
1,"Hopper","Giordano","HoGi","HoGi1"
2,"Harmony","Campbell","HaCa","Harmony7Campbell"
3,"Harmony","Clark","HaCl","Harmony4Clark"
4,"Holly","Collins","HoCo","Holly7Collins"
```

UPDATE st_person set cVName = "HaHa"WHERE cVName like „Ha“; Hier werden mehrere Datensätze gleichzeitig geändert

```
SELECT * FROM st_person;

"STPerson","cVName","cFName","cMName","cRName"
1,"Hopper","Giordano","HoGi","HoGi1"
2,"HaHa","Campbell","HaCa","Harmony7Campbell"
3,"HaHa","Clark","HaCl","Harmony4Clark"
4,"Holly","Collins","HoCo","Holly7Collins"
```

DELETE

Mittels dem Befehl *DELETE*werden Datensätze in der Datenbank gelöscht.

Einfache Darstellung Beim einfachsten DELETE haben wir es mit wenigen Schlüsselwörter zu tun.

```
DELETE 'Ziel'
  WHERE 'Where-Klausel'
```

Aber Vorsicht, ohne entsprechende *WHERE*-Klausel werden alle betroffenen Datensätze gelöscht. Daher gilt, fehlt die WHERE-Klausel, so werden **alle Datensätze** unwiderruflich gelöscht!

Beispiele zu DELETE

DELETE FROM st_person WHERE STPerson = 1; Bei *DELETE* und anderen Befehlen zur Erstellung von Daten, gibt es kein Ergebnismenge. Hier wird der Datensatz mit dem Wert 1 in der Spalte STPerson gelöscht.

```
SELECT * FROM st_person;

"STPerson","cVName","cFName","cMName","cRName"
1,"Hope","Giordano","HoGi","HoGi"
2,"Harmony","Campbell","HaCa","Harmony7Campbell"
3,"Harmony","Clark","HaCl","Harmony4Clark"
4,"Holly","Collins","HoCo","Holly7Collins"

DELETE FROM st_person WHERE STPerson = 1;
SELECT * FROM st_person;

"STPerson","cVName","cFName","cMName","cRName"
2,"Harmony","Campbell","HaCa","Harmony7Campbell"
3,"Harmony","Clark","HaCl","Harmony4Clark"
4,"Holly","Collins","HoCo","Holly7Collins"
```

DELETE st_person WHERE STPerson = 1; Bei *DELETE* und anderen Befehlen zur Erstellung von Daten, gibt es kein Ergebnismenge. Hier werden **alle** Datensätze in der Tabelle st_person gelöscht!

```
SELECT * FROM st_person;

"STPerson","cVName","cFName","cMName","cRName"
1,"Hope","Giordano","HoGi","HoGi"
2,"Harmony","Campbell","HaCa","Harmony7Campbell"
3,"Harmony","Clark","HaCl","Harmony4Clark"
4,"Holly","Collins","HoCo","Holly7Collins"

DELETE FROM st_person;
SELECT * FROM st_person;

"STPerson","cVName","cFName","cMName","cRName"
```

3.1.4 DCL Datenkontrollsprache

Die Datenkontrollsprache (Data Control Language = DCL[4]) umfasst die Sprachteile von Datenbanksprache, mit deren Hilfe man die Rechte vergibt, Wartungen durchführt. Es sind das Befehle wie *GRANT* und *REVOKE*.

[4] siehe auch http://de.wikipedia.org/wiki/Data_Control_Language