

# Datebanken

## *Was ist eigentlich eine Datenbank?*

Datenbanken, Datenhaltungsschicht und Datenbanksysteme (hier als Synonyme zu verstehen) finden viele unterschiedliche Anwendungsbereiche.

Datenbanken kann man sich vorstellen wie große Exceltabellen. Die Möglichkeit mit DDL & DML (Befehle mit einer bestimmten Syntax zur Manipulation von Daten wtc.) Daten auszugeben und zu ändern ist auf das Datenbankmanagementsystem zurückzuführen und erleichtert die Arbeit ungemein. Eine strukturierte Logik hinter einer übersichtlichen Eingabemaske macht die Verwendung einer Datenbank attraktiv. Daten können in großen Massen bequem ausgelesen, eingegeben und verändert werden.

*Infos zur Zusammenfassung:*

*Die folgende Zusammenfassung bietet einen groben Überblick über die Syntax von SQL Befehlen. Desweiteren wird die Begrifflichkeit einer Datenbank erläutert. Diese Zusammenfassung ist sehr komprimiert gehalten und beinhaltet Halbwissen. Deshalb rate ich, dass Sie nur als Nachschlagewerk für neugierige benutzt wird, die schon ein wenig Grundlagen in der Materie Datenbanken sammeln konnten.*

## **Konkrete Einsatzgebiete :** (Von der Nähe zur Distanz zum User geordnet)

Hinter dem Betriebssystem Windows und Mac OS befinden sind zahlreiche große Datenbanken mit unzähligen Tabellen. Angefangen von der Tabelle, welche die einzelnen Benutzerkonten des PCs speichert, über die Tabelle in der unser Verlauf im Browser gespeichert ist bis hin zur Verwaltung der Daten auf unseren Festplatten. Desweiteren braucht fast jedes Programm, das auf unseren PCs läuft eine Datenbank, Word greift z.B. auf eine Font-Datenbank zu, in der die verschiedenen Schriftarten gespeichert sind.

In einem Netzwerk können zentral Daten auf einem Server gespeichert werden, so z.B. zentrale Dateien wie Login-Daten in einem Hochschul- oder Firmennetzwerk.

Ein größeren Aufgabenbereich können Datenbanksysteme in großen Unternehmen abdecken. Im Krankenhaus zum Beispiel, speichern Sie die Schichtpläne von Ärzten, Patientenakten, Röntgen- und MRT Bilder, etc... von daher ist es in vielen Aufgabenbereichen essentiell, dass ein DBMS eine sehr gute Erreichbarkeit hat. Die Geschwindigkeit ist in diesem Fall zweitrangig.

Der Einsatz eines DBMS auf einem Server im Internet, mit mehreren hunderttausend Zugriffen / Tag wie z.B. bei großen Online-Kaufhäusern um deren Artikeldaten zu speichern stellt wiederum ganz andere Anforderungen. Hier ist die Geschwindigkeit ein wichtiges Bewertungskriterium.

## **Definition – SQL:** (stark gekürzt)

*SQL ist eine Sprache um in Datenbanken Tabellen und allgemeine Strukturen zu erstellen, Werte einzufügen, zu ändern oder zu löschen. Die Syntax von SQL ist simpel und logisch aufgebaut, findet teilweise Anlehnung an anderen, höheren Programmiersprachen und stützt sich genau wie diese, an die englische Sprache.*

## SQL Zusammenfassung

### SELECT tabelle123.attribut456

→ Wird Projektion genannt , gibt den Inhalt von „attribut456“ aus

Man kann auch nur schreiben:

### SELECT attribut456

→ Dann darf aber nur in einer der Tabellen, die hinter FROM stehen das Attribut mit dem Namen „attribut456“ vorhanden sein.

### SELECT DISTINCT attribut456

→ Sorgt dafür, dass keine doppelten Zeilen ausgegeben werden

### SELECT attribut456

### FROM tabelle123

→ Es wird nach dem attribut456 gesucht, und das nur in der Tabelle „tabelle123“

### Selektionsbedingung WHERE:

### SELECT attribut456

### WHERE attribut456 [Bedingung (= < >...)] WERT

### Bsp:

WHERE Haarfarbe = 'braun'

WHERE Vermögen >= 50000

WHERE Name = ‚Müller‘

WHERE Name LIKE 'M?ller'

WHERE Name LIKE 'M%'

WHERE Haarfarbe IN ('schwarz', 'braun', 'rot')

Selektionsbedingungen können wie beim vielen Programmiersprachen logisch mit AND verknüpft werden:

WHERE Haarfarbe = 'braun' AND Vermögen >= 50000

desweiteren ist eine Negation von Selektionsbedingungen möglich:

WHERE Haarfarbe = 'braun' AND NOT Vermögen >= 50000

→ Ergebnisrelation enthält nur Zeilen von braunhaarigen mit einem Gehalt unter 50000

**SELECT tabelle123.attribut456 AS attr**

- ➔ Umbenennung des Attributs „attribut456“ in den Namen „attr“  
Diesen Namen kann man von nun an weiter im SQL Befehl benutzen

Oder

**SELECT tab.attribut456 AS attr****FROM tabelle123 AS tab**

- ➔ Umbenennung der Tabelle “tabelle123” in den Namen “tab”  
Funktioniert auch rückwirkend in der SELECT Anweisung

**SELECT attribut456****WHERE attribut456 [Bedingung (= < >...)] WERT****ORDER BY attribut456 desc**

- ➔ ORDER BY = Sortierung
- ➔ ASC = aufsteigend
- ➔ DESC = absteigend

**UNION**

```
SELECT attribut123
FROM tabelle345
WHERE Bedingung = Wert
UNION
SELECT attribut678
FROM tabelle901
```

- ➔ Eine “Ergebnistabelle” in der die Attribute attribut123 und attribut678 in identischen Spalten stehen

**Tabelle erstellen**

```
CREATE TABLE tabelle123(
id INTEGER NOT NULL,
Name VARCHAR(40) NOT NULL
PRIMARY KEY (id)
CONSTRAINT FK_tab1_tab2
FOREIGN KEY (id)
REFERENCES tabelle2(nummer)
)
```

- ➔ Kann auch: id INTEGER NOT NULL AUTO\_INCREMENT  
Sein, damit bei jedem neuen Eintrag die id eins hochgezählt wird
- ➔ Constraints = Sicherstellung der Gültigkeit von Primär-/ Fremdschlüssel-Werten

**Tabelle löschen**

```
DROP TABLE tabelle123
```

**Tabelle ändern**

```
ALTER TABLE tabelle123
ADD attribut456 VARCHAR(10)
DROP attribut321
ADD PRIMARY KEY(attribut123)
```

➔ Attribut hinzufügen / löschen oder als Primärschlüssel deklarieren

**Werte Einfügen**

```
INSERT INTO tabelle123 (id, name, verdienst)
VALUES (1, Hans, 25000)
```

**Werte Ändern**

```
UPDATE tabelle123
SET   attribut123 = wert
SET   attribut456 = attribut456 + wert
WHERE id =1
```

**Löschen von Entitäten**

```
DELETE FROM tabelle123
WHERE id=1
```

Relationen werden meist mittels PK und FK verknüpft, also PK aus einer Tabelle mit dem FK aus einer anderen.

Das funktioniert mit **JOINS**.

**JOIN**

Zwei Tabellen werden „Verbunden“. Wir müssen die jeweiligen Attribute verbinden, zwischen denen eine Verbindung bestehen soll.

Z.B. Bei der Tabelle bei einer Tabelle namens „Kunde“ das Attribut „Kundennummer“ (PK) und bei der Tabelle „Kontaktadresse“ das Attribut „KdNR“ (FK) um zwei konkrete Namen zu nennen.

INNER JOIN: Stupid es verbinden der Tabellen, wenn eine der Ausgegebenen Zeilen auf einer Seite kein zugewiesenen Wert haben würde (Kunde existiert, eine Kontaktadresse ist jedoch nicht vorhanden) würde der normale INNER JOIN die Spalte nicht ausgeben.

**LEFT / RIGHT JOIN**

FROM KUNDE LEFT JOIN Kontaktadresse würden auch Kunden ohne Kontaktadresse ausgegeben werden

FROM KUNDE RIGHT JOIN Kontaktadresse würden auch Kontaktadressen ohne dazugehörigen Kunden ausgegeben werden

Bsp:

```
SELECT K.name
FROM Kunde AS K LEFT JOIN Kontaktadresse AS KA
ON K.KundenNr = KA.KundenNr
WHERE KA.Wohnort = 'Ulm'
```

➔ Namen der Kunden aus Ulm werden ausgegeben. KundenNr ist FK von Kontaktadresse und wurde mittels Constraint als ein solcher definiert. Das könnte so ausgesehen haben:

```
CREATE TABLE Kontaktadresse(
  AdressNr INTEGER NOT NULL,
  KundenNr INTEGER NOT NULL,
  Wohnort VARCHAR(40) NOT NULL
  PRIMARY KEY (AdressNr)
  CONSTRAINT FK_Kontakt_Kunde
  FOREIGN KEY (KundenNr)
  REFERENCES Kunde(KundenNr)
)
```

**Gruppierung**

Fasst gleiche Werte einer Spalte zusammen

Hilfreich bei folgendem Beispiel (mit mehreren Zeilen):

Diese Werte sind uninteressant | Diese Werte sollen später ausgegeben werden

Konzern_Umsatz				
KundenNr	Name	KonzernNr	Konzern	Umsatz
1000	Müller GmbH	1000	Müller GmbH	100.000
1001	Stahl GmbH	1000	Müller GmbH	80.000
1005	Maier AG	1005	Maier AG	30.000
1006	Abc KG	1005	Maier AG	150.000

=Gruppierung

=Summation

```
SELECT Konzern, KonzernNr, SUM(Umsatz) AS Umsatz
FROM Konzern_Umsatz
GROUP BY Konzern, KonzernNr
```

**Ergebnisrelation:**

KonzernNr	Konzern	Umsatz
1000	Müller GmbH	180.000
1005	Maier AG	180.000

- **SQL Aggregatfunktionen**

COUNT(<Spaltenname>) (zählt Zeilen in der Gruppe)

SUM(<Spaltenname>)

AVG(<Spaltenname>) (berechnet Mittelwert)

MIN, MAX(<Spaltenname>)

**WICHTIG:** Wenn man eine Aggregatsfunktion anwendet (s.o.) dann **muss** man immer alle Attribute, die in der SELECT Anweisung stehen auch in GROUP BY schreiben **außer** die Aggregatsfunktion selber

**Anomalien (sollten vermieden werden), deshalb am besten angewöhnen, Tabellen immer in die 3. Normalform zu bekommen.**

**Updateanomalie:** Name von einem Kunden ändert sich und ich muss es an mehr als **einer** Stelle in der Datenbank ändern

**Einfügeanomalien:** Ich füge einen neuen Mitarbeiter ein, der noch keine Personalnummer hat (PK)

**Löschanomalie:** Die Kundenadresse steht beim jeweiligen Kunden in der Kundentabelle, ich lösche einen Kunden und verliere dadurch nun auch das Wissen, dass Ulm die PLZ 89073 hat (wie Tabelle [B] auf der nächsten Seite)

Eine gutstrukturierte Datenbank wäre normalisiert und hätte eine eigene Tabelle:

PLZ	Ort
89073	Ulm
89077	Ulm
81349	Hintertuckenheim

Tabelle [A]

### Normalisierung

1. Normalform:

- atomar:

Jedes Attribut der Relation muss einen atomaren Wertebereich haben

Die Spalte Name sollte nicht „Max Mustermann“ enthalten. Für Vor- und Nachname sollte eine separate Spalte vorhanden sein

2. Normalform:

- 1. NF
- Alle Attribute, die nicht zum Primärschlüssel gehören sind von ihm voll abhängig.

3. Normalform:

- 2. NF
- Auf ein Nichtschlüsselattribut kann man nicht durch ein anderes Nichtschlüsselattribut schließen.

**Beispiel 2.NF:**

<u>Personalnummer</u>	Nachname	Wohnort	PLZ
1	Maier	Ulm	89073
2	Schmidt	Hintertuckenhausen	81729

Tabelle [B]

→ 2. NF, da die 3. NF verletzt ist. Aus der PLZ lässt sich der Wohnort erkennen.

**Beispiel 3.NF:**

<u>Personalnummer</u>	Nachname	<u>PLZ</u>	Wohnort	<u>Personalnummer</u>
1	Maier	89073	Ulm	1
2	Schmidt	81729	Hintertuckenhausen	2

Tabelle [C]

→ 3. NF, da die Attribute, die keine Schlüsselattribut sind voll vom Schlüsselattribut abhängig sind.

**Mehrschichtenarchitektur**

Präsentation: GUI (grafische Nurtzeroberfläche, Java SWT, ...), SAP Oberfläche (Umfangreiches Programm für unterschiedlichste Geschäftsanwendungen), Internetseite im Webbrowser (http-Client), etc...

→ Oberfläche zur Ein-oder Ausgabe von Daten

Business-Logik: Was steckt hinter der Präsentationsschicht? Was wird mit den Daten gemacht? Evtl wird mit ihnen gerechnet, oder anderweitig geändert bzw. manipuliert. (Beispiel: Anzahl und Menge wurde eingegeben, Business-Logik multipliziert die Werte).

→ Die Aufgabe der Business-Logik ist, die Daten die in der Präsentationsschicht eingegeben worden sind, zu verarbeiten und in eine Datenbank zu schreiben.

Datenhaltung: DBMS: Datenhaltung (Programm) + Daten  
Die Daten, die in der GUI eingegeben wurden und in der Business-Logik verarbeitet wurden und in die DaBa geschrieben wurden befinden sich nun hier. In der Datenhaltungsschicht

→ In der Realität liegen die Daten dann entweder in einem DBMS auf einem Server im Intranet (z.B. Logindaten für Mitarbeiter einer Firma), oder sind auf einem Webserver und speichern dort Daten für Nutzer aus dem Internet (z.B. Artikel von Onlinekaufhäusern).