Einführung in SQL

Dieses Material lehnt sich an das englischsprachige SQL-Tutorial des W3C an (Google: w3c sql school). Insbesondere kommen die meisten Beispiele von dort. Das W3C-Tutorial bietet die Möglichkeit, SQL-Anfragen auszuführen (SQL Try It) und einen SQL-Test zu durchzuführen (SQL Quiz).

Erste Annäherung: SQL ist eine standardisierte Sprache für den Zugriff und die Manipulation von Datenbanken.

SQL in Stichpunkten

- SQL steht für Structured Query Language
- SQL gestattet den Zugriff auf Datenbanken
- SQL ist eine durch das ANSI (s.u.) standardisierte Sprache
- SQL kann Anfragen an eine Datenbank effizient beantworten
- SQL kann Daten in einer Datenbank effizient finden
- SQL kann neue Sätze (Zeilen, Tupel, Records) in eine Datenbank einfügen
- SQL kann Sätze aus einer Datenbank löschen
- SQL kann Sätze in einer Datenbank ändern
- SQL nimmt für sich in Anspruch, einfach zu lernen zu sein

SQL ist ein Standard, aber ...

SQL ist eine durch das ANSI (American National Standards Institute, vergleichbar mit der Deutsche Industrie Norm DIN) standardisierte Sprache für den Zugriff auf und die Manipulation von Datenbanken. SQL Statements werden verwendet, um Daten in einer Datenbank zu finden und zu verändern. SQL ist verfügbar in Datenbankprogrammen wie MS Access, DB2, Informix, MS SQL Server, Oracle, Sybase, PostgreSQL, etc.

Unglücklicherweise gibt es viele verschiedene SQL-Dialekte von unterschiedlichen Anbietern, aber sie sollten sich alle an den ANSI-Standard halten, und sie sollten die wichtigsten Schlüsselworte in gleicher Weise unterstützen (wie z.B. SELECT, UPDATE, DELETE, INSERT, WHERE).

Die meisten Datenbankprogramme, die SQL unterstützen, unterstützen ihren eigenen proprietären Erweiterungen in Ergänzung zum SQL-Standard.

SQL-Versionen

Year	Name	Alias	Comments
1986	SQL-86	SQL-87	First published by ANSI. Ratified by ISO in 1987.
1989	SQL-89		Minor revision.
1992	SQL-92	SQL2	Major revision (ISO 9075).
1999	SQL:1999	SQL3	Added regular expression matching, recursive queries, triggers, non-scalar types and some object-oriented features. (The last two are somewhat controversial and not yet widely supported.)
2003	SQL:2003		Introduced XML-related features, *window functions*, standardized sequences and columns with auto-generated values (including identity-columns).
2006	SQL:2006		ISO/IEC 9075-14:2006 defines ways in which SQL can be used in conjunction with XML. It defines ways of importing and storing XML data in an SQL database, manipulating it within the database and publishing both XML and conventional SQL-data in XML form. In addition, it provides facilities that permit applications to integrate into their SQL code the use of XQuery, the XML Query Language published by the World Wide Web Consortium (W3C), to concurrently access ordinary SQL-data and XML documents.

Quelle: http://en.wikipedia.org/wiki/SQL

SQL-89: Level 1, Level 2, IEF (Integrity Enhancement Feature)

SQL-92: Entry Level, Intermediate Level, Full Level

SQL:1999: Core SQL Support, Enhanced SQL Support

SQL Datenbanktabellen

Eine Datenbank umfasst meistens eine oder mehrere Tabellen. Jede Tabelle wird identifiziert über ihren Namen (z.B. "Customers" oder "Orders"). Tabellen enhalten Sätze (Zeilen, Tupel, Records) mit Daten. Spalten (Attribute) gestatten die Strukturierung der Information.

Es folgt ein Beispiel einer Tabelle mit Namen "Persons":

LastName	FirstName	Address	City
Hansen	Ola	Timoteivn 10	Sandnes
Svendson	Tove	Borgvn 23	Sandnes

Pettersen	Kari	Storgt 20	Stavanger

Diese Tabelle enthält drei Sätze (jeder Satz repräsentiert eine Person) und vier Spalten (LastName, FirstName, Address, und City).

SQL-Queries

Mit SQL kann man Anfragen an eine Datenbank stellen und erhält eine Ergebnismenge.

Anfrage

```
SELECT LastName FROM Persons
```

Resultat

LastName
Hansen
Svendson
Pettersen

Detail zur Syntax: Einige Datenbanksysteme erwarten ein Semikolon am Ende eines SQL-Statements. In diesem Material wird kein Semikolon verwendet.

DML-Anteil (Data Manipulation Language) von SQL

SQL bietet Möglichkeiten zur Ausführung von Anfragen. Aber SQL enthält auch Sprachmittel zum Ändern, Einfügen und Löschen von Sätzen.

Die Anfrage- und Manipulationskommandos zusammengefasst bilden den DML-Teil (Data Manipulation Language) von SQL:

- SELECT - sucht Daten in Datenbanktabellen
- INSERT INTO - fügt Daten in eine Datenbanktabelle ein
- UPDATE - ändert Daten in einer Datenbanktabelle
- DELETE - löscht Daten aus einer Datenbanktabelle

DDL-Anteil (Data Definition Language) von SQL

Der DDL-Anteil (Data Definition Language) von SQL gestattet es,

die Strukturen von Datenbanktabellen zu erzeugen, zu ändern oder zu löschen. Weiter kann man Indexe (Unterstützung für Suchschlüssel und damit für den effizienten Zugriff) definieren, spezielle Verbindungen zwischen Tabellen spezifizieren (Fremdschlüssel) und Einschränkungen (Constraints) an Datenbanktabellen formulieren.

Die wichtigsten DDL-Statements in SQL sind:

- CREATE TABLE - legt die Struktur einer Datenbanktabelle fest
- ALTER TABLE - ändert die Struktur einer Datenbanktabelle
- DROP TABLE - löscht eine Datenbanktabelle
- CREATE INDEX - erzeugt einen Index (Unterstützung für Suchschlüssel)
- DROP INDEX - löscht einen Index

SQL SELECT Statement

Das SQL SELECT Statement

Das SELECT Statement wird verwendet, um Daten aus einer Tabelle (oder aus mehreren Tabellen) zu ermittlen. Das wiederum tabellenartige Resultat wird in einer Resultatstabelle (Ergebnismenge) präsentiert.

Syntax

```
SELECT column_name(s) FROM table_name
```

Die Überschrift Syntax bedeutet hier (wie auch an anderen Stellen in diesem Material), dass typische, aber nicht notwendigerweise alle syntaktischen Möglichkeiten angegeben sind (z.B. gibt es im SELECT Statement einen WHERE Teil).

SQL Statements sind nicht case-sensitive (Gross-/Kleinschreibung wird nicht beachtet). 'SELECT' bedeutet das gleiche wie 'select'. In den Daten werden aber Gross-und Kleinbuchstaben unterschieden.

Beispiel SQL SELECT

Um den Inhalt der Spalten mit Namen "LastName" und "FirstName" aus der Datenbanktabelle mit Namen "Persons" zu ermitteln, verwendet man das folgende SELECT Statement:

```
SELECT LastName, FirstName FROM Persons
```

Datenbanktabelle "Persons"

LastName	FirstName	Address	City
Hansen	Ola	Timoteivn 10	Sandnes
Svendson	Tove	Borgvn 23	Sandnes
Pettersen	Kari	Storgt 20	Stavanger

Resultat

LastName	FirstName
Hansen	Ola
Svendson	Tove
Pettersen	Kari

Selektion aller Spalten

Um alle Spalten aus der Tabelle "Persons" zu ermittlen, kann das Zeichen * statt der Spaltennamen verwendet werden:

```
SELECT * FROM Persons
```

Resultat

LastName	FirstName	Address	City
Hansen	Ola	Timoteivn 10	Sandnes
Svendson	Tove	Borgvn 23	Sandnes
Pettersen	Kari	Storgt 20	Stavanger

Die Resultatsmenge

Das Resultat einer SQL-Query wird in einer Resultatsmenge gespeichert. Die meisten Datenbanksysteme gestatten die Navigation in der Resultatsmenge mit Operationen wie: Move-To-First-Record, Get-Record-Content, Move-To-Next-Record, etc. Dies wird im Material über JDBC erklärt.

Das SELECT DISTINCT Statement

Das Schlüsselwort DISTINCT wird verwendet, wenn unterschiedliche Werte zurückgegeben werden sollen. D.h. in SQL muss man DISTINCT im SELECT Statement verwenden, wenn man unterschiedliche Werte

erhalten will:

Syntax

```
SELECT DISTINCT column_name(s)
FROM table_name
```

Beispiele zu DISTINCT

Um *alle* Werte der Spalte "Company" zu ermitteln, verwendet man ein SELECT Statement wie:

```
SELECT Company FROM Orders
```

"Orders"

Company	OrderNumber
Sega	3412
W3Schools	2312
Trio	4678
W3Schools	6798

Resultat:

Company
Sega
W3Schools
Trio
W3Schools

Man beachte, dass "W3Schools" zweimal aufgeführt wird.

Um *unterschiedliche* Werte aus der Spalte "Company" zu erhalten, benutzt man das SELECT DISTINCT Statement:

```
SELECT DISTINCT Company FROM Orders
```

Resultat

Company
Sega
W3Schools
Trio

Es werden Duplikate eliminiert und "W3Schools" wird nur einmal in der Resultatsmenge aufgeführt.

SQL WHERE-Klausel

Die WHERE-Klausel wird benutzt, um ein Selektionskriterium zu spezifizieren.

Die WHERE-Klausel

Um Daten unter Verwendung einer Bedingung aus einer Tabelle zu selektieren, kann die WHERE-Klausel zu einem SELECT Statement hinzugefügt werden.

Syntax

```
SELECT column FROM table
WHERE column operator value
```

In der WHERE-Klausel können u.a. die folgenden Operatoren benutzt werden:

Operator	Beschreibung
=	Gleich
<>	Ungleich
>	Grösser als
<	Kleiner als
>=	Grösser als oder gleich
<=	Kleiner als oder gleich
BETWEEN	Intervalsuche
LIKE	Suche nach einem Muster
IS NULL, IS NOT NULL	Test auf NULL-Wert

In einigen Versionen von SQL wird der Operator <> als != geschrieben.

Verwendung der WHERE-Klausel

Um Personen auszuwählen, die in der City "Sandnes" leben, wird folgende WHERE-Klausel zum SELECT Statement hinzugefügt:

```
SELECT * FROM Persons
WHERE City='Sandnes'
```

"Persons"

LastName	FirstName	Address	City	Year
Hansen	Ola	Timoteivn 10	Sandnes	1951
Svendson	Tove	Borgvn 23	Sandnes	1978
Svendson	Stale	Kaivn 18	Sandnes	1980
Pettersen	Kari	Storgt 20	Stavanger	1960

Resultat

LastName	FirstName	Address	City	Year
Hansen	Ola	Timoteivn 10	Sandnes	1951
Svendson	Tove	Borgvn 23	Sandnes	1978
Svendson	Stale	Kaivn 18	Sandnes	1980

Verwendung von Anführungszeichen

Man beachte die Anführungszeichen bei dem Auswahlwert in der Beispielanfrage.

SQL verwendet einfache Anführungszeichen, um Text zu kennzeichnen. Viele Datenbanksysteme akzeptieren auch doppelte Anführungszeichen. Numerische Werte werden nicht in Anführunsgzeichen eingeschlossen.

Für Textwerte

```
Syntaktisch korrekt:
SELECT * FROM Persons WHERE FirstName='Tove'

Syntaktisch falsch:
SELECT * FROM Persons WHERE FirstName=Tove
```

Für numerische Werte

```
Syntaktisch korrekt:
SELECT * FROM Persons WHERE Year>1965

Syntaktisch falsch:
SELECT * FROM Persons WHERE Year>'1965'
```

Die LIKE Bedingung

Die LIKE Bedingung wird verwendet, um ein Suchmuster für eine Spalte anzugeben.

Syntax

```
SELECT column FROM table
WHERE column LIKE pattern
```

Das Zeichen "%" kann als Wildcard (beliebige Buchstabenfolge in einem Muster) sowohl vor als auch nach einem Muster verwendet werden.

Verwendung von LIKE

Personen bei denen FirstName mit einem 'O' beginnt:

```
SELECT * FROM Persons
WHERE FirstName LIKE 'O%'
```

Personen bei denen FirstName mit einem 'a' endet:

```
SELECT * FROM Persons
WHERE FirstName LIKE '%a'
```

Personen bei denen FirstName das Muster 'la' enthält:

```
SELECT * FROM Persons
WHERE FirstName LIKE '%la%'
```

Das INSERT INTO Statement

Das INSERT INTO statement wird benutzt, um neue Sätze in eine Tabelle einzutragen.

Syntax

```
INSERT INTO table_name
VALUES (value1, value2,....)
```

Man kann auch angeben, in welche Spalten die Daten eingetragen werden sollen:

```
INSERT INTO table_name (column1, column2,...)
VALUES (value1, value2,....)
```

Einfügen eines neuen Satzes

"Persons"

LastName	FirstName	Address	City
Pettersen	Kari	Storgt 20	Stavanger

SQL Statement

```
INSERT INTO Persons
VALUES ('Hetland', 'Camilla', 'Hagabakka 24', 'Sandnes')
```

Resultat

LastName	FirstName	Address	City
Pettersen	Kari	Storgt 20	Stavanger
Hetland	Camilla	Hagabakka 24	Sandnes

Einfügen von Daten in angegebene Spalten

"Persons"

LastName	FirstName	Address	City
Pettersen	Kari	Storgt 20	Stavanger
Hetland	Camilla	Hagabakka 24	Sandnes

SQL Statement

```
INSERT INTO Persons (LastName, Address)
VALUES ('Rasmussen', 'Storgt 67')
```

Resultat

LastName	FirstName	Address	City
Pettersen	Kari	Storgt 20	Stavanger
Hetland	Camilla	Hagabakka 24	Sandnes
Rasmussen		Storgt 67	

Spalten, die nicht angegeben wurden, werden mit NULL-Werten gefüllt. Im obigen Beispiel sind diese Spalteneinträge leer dargestellt. Der NULL-Wert steht für 'Unbekannt' oder 'Keine Angabe'. Der NULL-Wert ist ungleich der numerischen Null (0), ungleich der leeren Zeichenkette ('') und ungleich einer Zeichenkette die nur aus Leerzeichen besteht (z.B. ' ').

Das Update Statement

Das UPDATE Statement wird verwendet, um Daten in einer Tabelle zu modifizieren.

Syntax

```
UPDATE table_name
SET column_name = new_value
WHERE column_name = some_value
```

Man beachte die unterschiedliche Bedeutung des Zeichens '='. Hier wird '=' als Zuweisung verwendet (auf der linken Seite ein Attribut, und auf der rechten Seite ein Wert bzw. ein Ausdruck). Weiter oben in SELECT Anweisungen wurde '=' als Gleichheitsabfrage verwendet (auf der linken und rechten Seite jeweils Ausdrücke).

Person

LastName	FirstName	Address	City
Nilsen	Fred	Kirkegt 56	Stavanger
Rasmussen		Storgt 67	

Update einer Spalte in einem Satz

Es soll die Spalte FirstName der Person mit LastName "Rasmussen" hinzugefügt werden:

```
UPDATE Person SET FirstName = 'Nina'
WHERE LastName = 'Rasmussen'
```

Resultat

LastName	FirstName	Address	City
Nilsen	Fred	Kirkegt 56	Stavanger
Rasmussen	Nina	Storgt 67	

Update verschiedener Spalten in einem Satz

Es sollen die Spalten Address und City verändert werden:

```
UPDATE Person
SET Address = 'Stien 12', City = 'Stavanger'
WHERE LastName = 'Rasmussen'
```

Resultat

LastName	FirstName	Address	City
Nilsen	Fred	Kirkegt 56	Stavanger
Rasmussen	Nina	Stien 12	Stavanger

SQL DELETE Statement

Das DELETE Statement

Das DELETE Statement wird benutzt, um Sätze in einer Tabelle zu löschen.

Syntax

```
DELETE FROM table_name
WHERE column_name = some_value
```

Person

LastName	FirstName	Address	City
Nilsen	Fred	Kirkegt 56	Stavanger
Rasmussen	Nina	Stien 12	Stavanger

Löschen eines Satzes

Alle Sätze mit "LastName = Rasmussen" werden gelöscht:

```
DELETE FROM Person WHERE LastName = 'Rasmussen'
```

Resultat

LastName	FirstName	Address	City
Nilsen	Fred	Kirkegt 56	Stavanger

Löschen aller Sätze

Man kann auch alle Sätze einer Tabelle löschen ohne die Tabelle selbst zu entfernen. Das bedeutet, dass die Tabellenstruktur, die Attribute und die Struktur der Indexe (s.u.) erhalten bleiben.

```
DELETE FROM table_name

oder

DELETE * FROM table_name
```

SQL ORDER BY

Das ORDER BY Schlüsselwort benutzt man, um das Resultat zu sortieren.

Sortierung von Sätzen

Die ORDER BY-Klausel dient damit zur Sortierung der Sätze nach

unterschiedlichen Kriterien.

Orders

Company	OrderNumber
Sega	3412
ABC Shop	5678
W3Schools	2312
W3Schools	6798

Beispiel

Um die alphabetische Reihenfolge für die Spalte Company zu erzielen:

```
SELECT Company, OrderNumber FROM Orders
ORDER BY Company
```

Resultat

Company	OrderNumber
ABC Shop	5678
Sega	3412
W3Schools	6798
W3Schools	2312

Beispiel

Um die alphabetische Reihenfolge für die Spalte Company und die numerische Reihenfolge für die Spalte OrderNumber zu erzielen:

```
SELECT Company, OrderNumber FROM Orders
ORDER BY Company, OrderNumber
```

```
SELECT Company, OrderNumber FROM Orders
ORDER BY 1, 2
```

Resultat

Company	OrderNumber
ABC Shop	5678
Sega	3412

W3Schools	2312
W3Schools	6798

Beispiel

Um die umgekehrte alphabetische Reihenfolge für die Spalte Company zu erzielen verwendet man DESC[ENDING] = fallend. Es kann auch ASC[ENDING] = steigend angegeben werden:

```
SELECT Company, OrderNumber FROM Orders
ORDER BY Company DESC
```

Resultat:

Company	OrderNumber
W3Schools	6798
W3Schools	2312
Sega	3412
ABC Shop	5678

Beispiel

Um die umgekehrte alphabetische Reihenfolge für die Spalte Company und die numerische Reihenfolge für OrderNumber zu erzielen:

```
SELECT Company, OrderNumber FROM Orders
ORDER BY Company DESC, OrderNumber ASC
```

Resultat

Company	OrderNumber
W3Schools	2312
W3Schools	6798
Sega	3412
ABC Shop	5678

Man beachte, dass es zwei Zeilen mit gleichem Wert für Company gibt (W3Schools). Die Notwendigkeit zur Angabe einer Sortierung für die zweite Spalte besteht nur, wenn es mehrere Zeilen mit gleichem Company-Wert gibt.

SQL AND und OR

AND und OR

AND und OR verbinden zwei oder mehr Bedingungen in der WHERE-Klausel.

Der AND Operator gibt eine Zeile aus, falls beide angegebenen Bedingungungen zutreffen. Der OR Operator gibt eine Zeile aus, falls eine der angegebenen Bedingungen zutrifft.

Gegebene Tabelle

LastName	FirstName	Address	City
Hansen	Ola	Timoteivn 10	Sandnes
Svendson	Tove	Borgvn 23	Sandnes
Svendson	Stephen	Kaivn 18	Sandnes

Beispiel

```
SELECT * FROM Persons
WHERE FirstName='Tove'
AND LastName='Svendson'
```

Resultat

LastName	FirstName	Address	City
Svendson	Tove	Borgvn 23	Sandnes

Beispiel

```
SELECT * FROM Persons
WHERE firstname='Tove'
OR lastname='Svendson'
```

Resultat:

LastName	FirstName	Address	City
Svendson	Tove	Borgvn 23	Sandnes
Svendson	Stephen	Kaivn 18	Sandnes

Beispiel

Kombination von AND und OR (Verwendung von Klammern um komplexe Ausdrücke zu bilden):

```
SELECT * FROM Persons
WHERE (FirstName='Tove' OR FirstName='Stephen')
AND LastName='Svendson'
```

Resultat

LastName	FirstName	Address	City
Svendson	Tove	Borgvn 23	Sandnes
Svendson	Stephen	Kaivn 18	Sandnes

SQL IN

IN

Der IN Operator kann verwendet werden, wenn man mehrere Auswahlwerte einer Spalte genau kennt.

```
SELECT column_name FROM table_name
WHERE column_name IN (value1,value2,..)
```

Beispieltabelle

LastName	FirstName	Address	City
Hansen	Ola	Timoteivn 10	Sandnes
Nordmann	Anna	Neset 18	Sandnes
Pettersen	Kari	Storgt 20	Stavanger
Svendson	Tove	Borgvn 23	Sandnes

Beispiel

Aus Persons alle Sätze mit LastName gleich "Hansen" oder "Pettersen":


```
SELECT * FROM Persons
WHERE LastName IN ('Hansen','Pettersen')
```

Resultat

LastName	FirstName	Address	City
Hansen	Ola	Timoteivn 10	Sandnes
Pettersen	Kari	Storgt 20	Stavanger

SQL BETWEEN

BETWEEN

Der BETWEEN Operator bestimmt einen Datenbereich zwischen zwei Werten. Die Werte können Zahlen, Text oder Datumsangaben sein.

```
SELECT column_name FROM table_name
WHERE column_name
BETWEEN value1 AND value2
```

Beispieltabelle

LastName	FirstName	Address	City
Hansen	Ola	Timoteivn 10	Sandnes
Nordmann	Anna	Neset 18	Sandnes
Pettersen	Kari	Storgt 20	Stavanger
Svendson	Tove	Borgvn 23	Sandnes

Beispiel

Alle Zeilen bei denen Name alphabetisch zwischen "Hansen" und "Pettersen" liegt:

```
SELECT * FROM Persons
WHERE LastName BETWEEN 'Hansen' AND 'Pettersen'
```

Resultat

LastName	FirstName	Address	City
Hansen	Ola	Timoteivn 10	Sandnes
Nordmann	Anna	Neset 18	Sandnes

Vorsicht: Der BETWEEN Operator wird von Datenbanksystem zu Datenbanksystem unterschiedlich behandelt. Die Bedeutung kann sein:

```
'Hansen' <= LastName AND LastName <= 'Pettersen'

'Hansen' <= LastName AND LastName < 'Pettersen'

'Hansen' < LastName AND LastName < 'Pettersen'
```

Man vergewissere sich vor Verwendung wie der BETWEEN Operator in dem jeweiligen System arbeitet. Der SQL Standard fordert bei beiden Grenzen den Vergleich '<='.

Beispiel

Der NOT Operator ist ein allgemeiner boolescher Operator der den Wahrheitswert einer Aussage umkehrt. NOT kann hier verwendet werden, um einen Bereich auszuschliessen:

```
SELECT * FROM Persons WHERE LastName
NOT BETWEEN 'Hansen' AND 'Pettersen'
```

Resultat

LastName	FirstName	Address	City
Pettersen	Kari	Storgt 20	Stavanger
Svendson	Tove	Borgvn 23	Sandnes

SQL Alias

In SQL gibt es Alias-Angaben für Spalten und Tabellen.

Spalten Alias

Syntax

```
SELECT column AS column_alias FROM table
```

Tabellen Alias

Syntax

```
SELECT column FROM table AS table_alias
```

Beispiel

Persons

LastName	FirstName	Address	City
Hansen	Ola	Timoteivn 10	Sandnes
Svendson	Tove	Borgvn 23	Sandnes
Pettersen	Kari	Storgt 20	Stavanger

SQL

```
SELECT LastName AS FamilyName, GivenName AS Name
FROM Persons
```

Resultat

FamilyName	GivenName
Hansen	Ola
Svendson	Tove
Pettersen	Kari

Beispiel

Persons

LastName	FirstName	Address	City
Hansen	Ola	Timoteivn 10	Sandnes
Svendson	Tove	Borgvn 23	Sandnes
Pettersen	Kari	Storgt 20	Stavanger

SQL

```
SELECT LastName, FirstName
```

```
FROM Persons AS Employees
```

Resultat

Tabelle Employees

LastName	FirstName
Hansen	Ola
Svendson	Tove
Pettersen	Kari

Tabellen-Aliase werden benötigt, wenn eine Tabelle in einer Anfrage mehrmals verwendet wird.

SQL JOIN

Joins und Schlüssel

Wenn man mehrere Tabellen benötigt, kann eine Anfrage mit einem Join (Verbund) formuliert werden.

In der unten angegebenen Tabelle "Employees" soll die Spalte "Employee_ID" der Primärschlüssel sein. Das heisst, dass zwei Sätze (Zeilen) nicht den gleichen Wert für Employee_ID haben können. Mit Employee_ID als Schlüssel könnten auch zwei unterschiedliche Personen mit gleichen Namen auseinander gehalten werden.

In den Beispieltabellen unten gilt:

- **Die Spalte "Employee_ID" ist Primärschlüssel in "Employees"**
- **Die Spalte "Prod_ID" ist Primärschlüssel in "Orders"**
- **Die Spalte "Employee_ID" in "Orders" verweist durch die Verwendung des Primärschlüssels aus "Employees" auf eine Person in "Employees"**

Employees

Employee_ID	Name
01	Hansen, Ola
02	Svendson, Tove
03	Svendson, Stephen
04	Pettersen, Kari

Orders

Prod_ID	Product	Employee_ID
234	Printer	01
657	Table	03
865	Chair	03

Zugriff auf zwei Tabellen

Beispiel

Wer hat welches Produkt bestellt?

```
SELECT Employees.Name, Orders.Product
FROM Employees, Orders
WHERE Employees.Employee_ID=Orders.Employee_ID
```

```
SELECT E.Name, O.Product
FROM Employees E, Orders O
WHERE E.Employee_ID=O.Employee_ID
```

Resultat

Name	Product
Hansen, Ola	Printer
Svendson, Stephen	Table
Svendson, Stephen	Chair

Beispiel

Wer hat einen Printer bestellt?

```
SELECT Employees.Name
FROM Employees, Orders
WHERE Employees.Employee_ID=Orders.Employee_ID
AND Orders.Product='Printer'
```

Resultat

Name
Hansen, Ola

Verwendung von Joins

Alternativ kann das Schlüsselwort JOIN benutzt werden:

Beispiel INNER JOIN

Syntax

```
SELECT field1, field2, field3
FROM first_table
INNER JOIN second_table
ON first_table.key = second_table.reference (or other columns)
```

Wer hat welches Produkt bestellt?

```
SELECT Employees.Name, Orders.Product
FROM Employees
INNER JOIN Orders
ON Employees.Employee_ID=Orders.Employee_ID
```

Der INNER JOIN gibt Angaben zurück die zueinander passen, d.h. im Employee_ID übereinstimmen. Falls Sätze aus Employees keine Entsprechung in Orders haben, erscheinen sie nicht im Ergebnis.

Resultat

Name	Product
Hansen, Ola	Printer
Svendson, Stephen	Table
Svendson, Stephen	Chair

Beispiel LEFT JOIN

Syntax

```
SELECT field1, field2, field3
FROM first_table
LEFT JOIN second_table
ON first_table.keyfield = second_table.foreign_keyfield
```

Ermittle alle Employees (Angestellten) und ihre Order (Bestellungen).

```
SELECT Employees.Name, Orders.Product
FROM Employees
LEFT JOIN Orders
ON Employees.Employee_ID=Orders.Employee_ID
```


Der LEFT JOIN gibt alle Sätze aus der linken Tabelle (Employees) zurück, auch wenn es keinen Entsprechung in der rechten Tabelle (Orders) gibt. Falls es Sätze in Employees gibt, die keine Order haben, werden sie auch ins Ergebnis aufgenommen, wobei fehlende Werte mit NULL-Werten aufgefüllt werden.

Resultat

Name	Product
Hansen, Ola	Printer
Svendson, Tove	
Svendson, Stephen	Table
Svendson, Stephen	Chair
Pettersen, Kari	

Beispiel RIGHT JOIN

Syntax

```
SELECT field1, field2, field3
FROM first_table
RIGHT JOIN second_table
ON first_table.keyfield = second_table.foreign_keyfield
```

Ermittle alle Orders (Bestellungen) und ihre Employees (Besteller).

```
SELECT Employees.Name, Orders.Product
FROM Employees
RIGHT JOIN Orders
ON Employees.Employee_ID=Orders.Employee_ID
```

Der RIGHT JOIN gibt alle Sätze aus der rechten Tabelle (Orders) zurück, auch wenn es keinen Entsprechung in der linken Tabelle (Employees) gibt. Falls es Sätze in Orders gibt, die keinen Employee haben, werden sie auch ins Ergebnis aufgenommen, wobei fehlende Werte mit NULL-Werten aufgefüllt werden.

Resultat

Name	Product
Hansen, Ola	Printer
Svendson, Stephen	Table
Svendson, Stephen	Chair

Beispiel

Wer hat einen Printer bestellt?

```
SELECT Employees.Name
FROM Employees
INNER JOIN Orders
ON Employees.Employee_ID=Orders.Employee_ID
WHERE Orders.Product = 'Printer'
```

Result

Name
Hansen, Ola

SQL UNION und UNION ALL

UNION

Ein UNION Ausdruck wird verwendet, um ähnliche Informationen aus zwei Tabellen zu ermitteln. Die ausgewählten Spalten müssen den gleichen Datentyp besitzen. Die Anzahl der ausgewählten Spalten muss übereinstimmen.

Mit UNION werden unterschiedliche Werte zurückgegeben.

```
SQL Statement 1
UNION
SQL Statement 2
```

Employees_Norway

E_ID	E_Name
01	Hansen, Ola
02	Svendson, Tove
03	Svendson, Stephen
04	Pettersen, Kari

Employees_USA

E_ID	E_Name
01	Turner, Sally
02	Kent, Clark
03	Svendson, Stephen
04	Scott, Stephen

Verwendung von UNION

Beispiel

Ermittle alle unterschiedlichen Employee-Namen aus Norwegen und den USA:

```
SELECT E_Name FROM Employees_Norway
UNION
SELECT E_Name FROM Employees_USA
```

Resultat

E_Name
Hansen, Ola
Svendson, Tove
Svendson, Stephen
Pettersen, Kari
Turner, Sally
Kent, Clark
Scott, Stephen

UNION in dieser Form liefert keine Duplikate. Im Beispiel gibt es einen Employee mit gleichem Namen in Norwegen und den USA (Svendson, Stephen), aber der Name wird nur einmal genannt. UNION in dieser Form liefert nur unterschiedliche Werte.

UNION ALL

UNION ALL arbeitet wie UNION mit dem Unterschied, dass hier alle Duplikate geliefert werden.

```
SQL Statement 1
UNION ALL
SQL Statement 2
```

Verwendung von UNION ALL

Beispiel

Ermittle alle Employees aus Norwegen und den USA inklusive der

Duplikate:

```
SELECT E_Name FROM Employees_Norway
UNION ALL
SELECT E_Name FROM Employees_USA
```

Resultat

E_Name
Hansen, Ola
Svendson, Tove
Svendson, Stephen
Pettersen, Kari
Turner, Sally
Kent, Clark
Svendson, Stephen
Scott, Stephen

Man beachte, dass 'Svendson, Stephen' zweimal aufgeführt wird.

Übersicht zu 'SELECT/UNION versus Duplikate'

	Ohne Duplikate	Mit Duplikaten
SELECT		X
SELECT DISTINCT	X	
UNION	X	
UNION ALL		X

SQL CREATE DATABASE, TABLE, und INDEX

Erzeugen einer Datenbank

Erzeugen einer Datenbank:

```
CREATE DATABASE database_name
```

In einer Datenbank kann es mehrere Datenbanktabellen geben.

Erzeugen einer Tabelle

Erzeugen einer Tabelle in einer Datenbank:

```
CREATE TABLE table_name (
column_name1 data_type,
column_name2 data_type,
... )
```

Beispiel

Erzeugen einer Tabelle mit Namen "Person" und den Spalten "LastName", "FirstName", "Address", und "Age":

```
CREATE TABLE Person (
LastName varchar,
FirstName varchar,
Address varchar,
Age int)
```

Angabe einer Maximallänge für einige Spalten:

```
CREATE TABLE Person (
LastName varchar(30),
FirstName varchar,
Address varchar,
Age int(3) )
```

Der Datentyp beschreibt die Art der Spaltenwerte. Die geläufigsten Datentypen in SQL sind:

Data Type	Beschreibung
integer(size) int(size) smallint(size) tinyint(size)	Für ganzzahlige numerische Werte. Anzahl der Ziffern in Klammern.
decimal(size,d) numeric(size,d)	Für numerische Werte mit Kommastellen. size ist die Gesamtanzahl der Ziffern, d die Anzahl der Kommastellen.
char(size)	Zeichenketten fester Länge. size ist die Anzahl der Zeichen.
varchar(size)	Zeichenketten variabler Länge. size ist die Maximalanzahl der Zeichen.
date	Datumsangaben im Format yyyymmdd.
time	Zeitangaben im Format hhmmss.

CREATE INDEX

Indexe können auf existierenden Tabellen erzeugt werden. Sie dienen dazu Sätze schneller und effizienter zu finden. Man kann mehr als einen Index für eine Tabelle angeben. Ein Index kann sich auf mehrere Attribute beziehen und einen Namen haben. Benutzer manipulieren und sehen Indexe nicht direkt, sie bemerken nur die effizientere Bearbeitung ihrer Anfragen.

Das Ändern einer Tabelle mit Index ist aufwändiger als bei einer Tabelle ohne Index, da nicht nur die eigentlichen Daten sondern auch der Index verändert werden muss. Indexe sollten daher nur für Spalten definiert werden, die oft in Anfragen verwendet werden.

UNIQUE INDEX: In einem mit UNIQUE erzeugten Index können zwei Spaltenwerte (zwei Kombinationen von Spaltenwerten) nicht den gleichen Indexwert haben.

```
CREATE UNIQUE INDEX index_name
ON table_name (column_name)
```

"column_name" gibt die Spalte an die indiziert werden soll.

Einfacher Index

Wenn UNIQUE nicht angegeben wird, sind Duplikate im Index erlaubt. UNIQUE wird typischerweise für Primärschlüssel angegeben.

```
CREATE INDEX index_name
ON table_name (column_name)
```

"column_name" gibt die Spalte an die indiziert werden soll.

Beispiel

Erzeuge einen einfachen Index mit der Bezeichnung "PersonIndex" auf der Spalte LastName der Tabelle Person:

```
CREATE INDEX PersonIndex
ON Person (LastName)
```

Indexe in absteigender Reihenfolge können mit DESC nach dem Spaltenname erzeugt werden:

```
CREATE INDEX PersonIndex
ON Person (LastName DESC)
```


Indexe auf Kombinationen von Attributen können durch eine Spaltenliste erzeugt werden:

```
CREATE INDEX PersonIndex
ON Person (LastName, FirstName)
```

Beispiele für Indexe

Persons	Tupelidentifier	LastName	FirstName	Address	City	Year
	t42	Hansen	Ola	Timoteivn 10	Sandnes	1951
	t44	Svendson	Tove	Borgvn 23	Sandnes	1978
	t46	Svendson	Stale	Kaivn 18	Sandnes	1980
	t48	Pettersen	Kari	Storgt 20	Stavanger	1960

INDEX(LastName)	Indexwert	Referenzierte Tupelmenge
	Hansen	{t42}
	Pettersen	{t48}
	Svendson	{t44, t46}

UNIQUE INDEX(LastName ASC, FirstName DESC)	Indexwert	Referenzierte Tupelmenge
	(Hansen,Ola)	{t42}
	(Pettersen,Kari)	{t48}
	(Svendson,Tove)	{t44}
	(Svendson,Stale)	{t46}

INDEX(City)	Indexwert	Referenzierte Tupelmenge
	Sandnes	{t42,t44,t46}
	Stavanger	{t48}

SQL DROP INDEX, TABLE und DATABASE

DROP INDEX

Ein existierender Index kann mit DROP INDEX entfernt werden.

Syntax in Microsoft SQLJet und Microsoft Access:

```
DROP INDEX index_name ON table_name
```

Syntax in MS SQL Server:

```
DROP INDEX table_name.index_name
```

Syntax in IBM DB2 und Oracle:

```
DROP INDEX index_name
```

Syntax in MySQL:

```
ALTER TABLE table_name DROP INDEX index_name
```

Die Details der syntaktischen Unterschiede sind nicht wichtig. Dies ist ein typisches Beispiel für die unterschiedliche Darstellung von Statements in unterschiedlichen Datenbanksystemen.

Löschen einer Tabelle oder einer Datenbank

Löschen einer Tabelle (inklusive Löschen der Tabellenstruktur, der Attribute und der Indexe):

```
DROP TABLE table_name
```

Löschen einer Datenbank:

```
DROP DATABASE database_name
```

Löschen von Daten

TRUNCATE TABLE löscht die Datensätze, aber nicht die Tabellenstruktur:


```
TRUNCATE TABLE table_name
```

SQL ALTER TABLE

ALTER TABLE

ALTER TABLE wird verwendet, um Spalten in einer Tabelle hinzuzufügen oder zu löschen.

```
ALTER TABLE table_name
ADD column_name datatype

ALTER TABLE table_name
DROP COLUMN column_name
```

Nicht alle Datenbanksystem gestatten die Löschung von Spalten (DROP COLUMN column_name).

Person

LastName	FirstName	Address
Pettersen	Kari	Storgt 20

Beispiel

Spalte "City" in "Person" hinzufügen:

```
ALTER TABLE Person ADD City varchar(30)
```

Resultat:

LastName	FirstName	Address	City
Pettersen	Kari	Storgt 20	

Fehlende Werte werden hier mit NULL-Werten aufgefüllt.

Beispiel

Löschen von "Address" in "Person":

```
ALTER TABLE Person DROP COLUMN Address
```

Resultat

LastName	FirstName	City
Pettersen	Kari	

SQL-Funktionen

SQL hat eine Reihe von Builtin-Funktionen für Berechnungen.

Syntax

Die Syntax gestattet die Verwendung z.B. in der SELECT-Klausel.

```
SELECT function(column) FROM table
```

Arten von Funktionen

In SQL gibt es:

- **Aggregationsfunktionen**
- **Skalare Funktionen**

Aggregationsfunktionen

Aggregationsfunktionen arbeiten auf Kollektionen von Werten, geben aber einen einzelnen Wert zurück.

Aggregationsfunktionen werden oft in Kombination mit der GROUP BY-Klausel (s.u.) verwendet.

"Persons"

Name	Age
Hansen, Ola	34
Svendson, Tove	45
Pettersen, Kari	19

SQL AVG

AVG gibt den Durchschnitt einer Spalte zurück. NULL-Werte werden nicht berücksichtigt.

Syntax

```
SELECT AVG(column) FROM table
```

Beispiel

```
SELECT AVG(Age) FROM Persons
```

Resultat

```
32.67
```

Beispiel

```
SELECT AVG(Age) FROM Persons WHERE Age>20
```

Resultat

```
39.5
```

SQL COUNT

COUNT(column) gibt die Anzahl von Sätzen zurück. NULL-Werte werden nicht berücksichtigt.

Syntax

```
SELECT COUNT(column) FROM table
```

Beispiel

"Persons"

Name	Age
Hansen, Ola	34
Svendson, Tove	45
Pettersen, Kari	

```
SELECT COUNT(Age) FROM Persons
```

Resultat

```
2
```

SQL COUNT(*)

COUNT(*) gibt die Anzahl der Sätze zurück.

Syntax

```
SELECT COUNT(*) FROM table
```

Beispiel

"Persons"

Name	Age
Hansen, Ola	34
Svendson, Tove	45
Pettersen, Kari	19

```
SELECT COUNT(*) FROM Persons
```

Resultat

```
3
```

Beispiel

```
SELECT COUNT(*) FROM Persons WHERE Age>20
```


Resultat

```
2
```

SQL COUNT DISTINCT

Dieses Beispiel funktioniert in ORACLE und Microsoft SQL aber nicht unter Microsoft Access.

DISTINCT und COUNT können zusammen verwendet werden, um die Anzahl der unterschiedlichen Ergebnisse zu ermitteln.

Syntax

```
SELECT COUNT(DISTINCT column(s)) FROM table
```

Beispiel

"Orders"

Company	OrderNumber
Sega	3412
W3Schools	2312
Trio	4678
W3Schools	6798

Beispiel

```
SELECT COUNT(Company) FROM Orders
```

Resultat

```
4
```

Beispiel

```
SELECT COUNT(DISTINCT Company) FROM Orders
```

Resultat

```
3
```

SQL MAX

MAX gibt den höchsten Wert einer Spalte zurück. NULL-Werte werden nicht berücksichtigt.

Syntax

```
SELECT MAX(column) FROM table
```

Beispiel

```
SELECT MAX(Age) FROM Persons
```

Resultat

```
45
```

MIN und MAX arbeiten auch mit Textspalten und berücksichtigen die alphabetische Reihenfolge.

SQL MIN

MIN gibt den niedrigsten Wert in einer Spalte zurück. NULL-Werte werden nicht berücksichtigt.

Syntax

```
SELECT MIN(column) FROM table
```

Beispiel

```
SELECT MIN(Age) FROM Persons
```


Resultat:

```
19
```

SQL SUM

SUM gibt die Summe der Spaltenwerte zurück. NULL-Werte werden nicht berücksichtigt.

Syntax

```
SELECT SUM(column) FROM table
```

Beispiel

```
SELECT SUM(Age) FROM Persons
```

Resultat

```
98
```

Beispiel

```
SELECT SUM(Age) FROM Persons WHERE Age>20
```

Resultat

```
79
```

Aggregationsfunktionen (Auswahl)

Funktion	Beschreibung
AVG(column)	Durchschnitt
COUNT(column)	Anzahl in der Spalte
COUNT(DISTINCT column)	Anzahl der verschiedenen Werte in der Spalte

COUNT(*)	Anzahl der Sätze
MAX(column)	Maximum
MIN(column)	Minimum
SUM(column)	Summe

Skalare Funktionen (Auswahl)

Skalare Funktionen arbeiten auf einzelnen Werten und geben einzelne Werte zurück.

Funktion	Beschreibung
UCASE(c)	Grossbuchstaben
LCASE(c)	Kleinbuchstaben
MID(c,start[,end])	Teilzeichenkette
LEN(c)	Länge
INSTR(c,char)	Position eines Zeichens
LEFT(c,number_of_char)	Linke Teilzeichenkette
RIGHT(c,number_of_char)	Rechte Teilzeichenkette
ROUND(c,decimals)	Runden
MOD(x,y)	Modulo
NOW()	Aktuelles Datum

SQL GROUP BY und HAVING

Aggregationsfunktionen (wie SUM) beziehen sich oft auf mehrere Sätzen mit gleichen Eigenschaften.

GROUP BY

Die Klausel GROUP BY wurde in SQL aufgenommen, da Aggregationsfunktionen (wie z.B. SUM) einen einzigen Aggregatwert zurückgeben, die Berechnung aber mit den bisherigen Möglichkeiten für die Gesamttabelle geschieht. Mit GROUP BY kann man eine Tabelle in Gruppen einteilen und Aggregationsfunktionen dann nur in den einzelnen Gruppen auswerten.

Syntax für GROUP BY

```
SELECT column, SUM(column) FROM table GROUP BY column
```


GROUP BY Beispiel

Tabelle "Sales"

Company	Amount
W3Schools	5500
IBM	4500
W3Schools	7100

SQL Anfrage

```
SELECT Company, SUM(Amount) FROM Sales
```

Resultat

Company	SUM(Amount)
W3Schools	17100
IBM	17100
W3Schools	17100

Mit der GROUP BY-Klausel können die Summenangaben gezielter gestaltet werden:

```
SELECT Company, SUM(Amount) FROM Sales
GROUP BY Company
```

Resultat:

Company	SUM(Amount)
W3Schools	12600
IBM	4500

HAVING

HAVING wurde eingeführt, um Selektionen auf den Gruppen (und nicht nur auf Sätzen wie in der WHERE-Klausel) zu ermöglichen.

Syntax für HAVING:

```
SELECT column, SUM(column) FROM table
GROUP BY column
HAVING SUM(column) condition value
```

"Sales" Tabelle

Company	Amount
W3Schools	5500
IBM	4500
W3Schools	7100

SQL

```
SELECT Company, SUM(Amount) FROM Sales
GROUP BY Company
HAVING SUM(Amount)>10000
```

Resultat

Company	SUM(Amount)
W3Schools	12600

Anfragen mit GROUP BY haben Einschränkungen hinsichtlich der syntaktisch erlaubten Ausdrücke in der SELECT- und HAVING-Klausel. Ein Ausdruck dort muss pro Gruppe einen Wert liefern. Typischerweise verwendet man als Ausdrücke Gruppierungsattribute oder Aggregationsfunktionen angewendet auf Nicht-Gruppierungsattribute.

SQL SELECT INTO Statement

Das SELECT INTO Statement

Das SELECT INTO Statement wird oft verwendet, um Kopien von Tabellen zu erstellen.

Syntax

```
SELECT column_name(s) INTO newtable [IN externalDatenbank]
FROM source
```

Erstellen einer Backup-Kopie

Kopie von "Persons":


```
SELECT * INTO PersonsBackup
FROM Persons
```

Die IN-Klausel kann zur Angabe einer anderen Datenbank verwendet werden.

```
SELECT Persons.* INTO Persons IN 'Backup.mdb'
FROM Persons
```

Man kann auch nur eine Teilmenge der vorhandenen Attribute spezifizieren:

```
SELECT LastName, FirstName INTO PersonsBackup
FROM Persons
```

Ferner kann man mit einer WHERE-Klausel eine Bedingung formulieren.

```
SELECT LastName,Firstname INTO PersonsBackup
FROM Persons
WHERE City='Sandnes'
```

Ferner ist es möglich, aus mehreren Tabellen die Daten zu beziehen:

```
SELECT Employees.Name, Orders.Product
INTO Empl_Ord_Backup
FROM Employees
INNER JOIN Orders
ON Employees.Employee_ID=Orders.Employee_ID
```

SQL CREATE VIEW Statement

Eine Sicht (View) ist eine virtuelle Tabelle, deren Inhalt sich als Ergebnismenge einer SELECT-Anfrage ergibt.

Sichten (Views)

Eine Sicht enthält Spalten und Sätze wie eine echte Tabelle. Die Spalten in einer Sicht können Spalten in einer echten Tabelle

entsprechen. Sichten können mit beliebigen Anfragen (inklusive komplexer WHERE-Klauseln und Joins) definiert werden. Bei der Benutzung der Sicht erscheint es dem Benutzer so, als kämen alle Spalten aus einer einzigen Tabelle.

Die Datenbankstruktur der echten Tabellen wird nicht durch die Einrichtung einer Sicht nicht verändert oder beeinflusst.

Syntax

```
CREATE VIEW view_name AS
SELECT column_name(s)
FROM table_name
WHERE Condition
```

Die Datenbank speichert die Sätze aus der Sicht nicht explizit. Die Datenbank erzeugt die Daten mit Hilfe des zugrundeliegenden SELECT Statements jedesmal wenn die Sicht angesprochen wird.

Verwendung von Views

Eine Sicht kann wie eine echte Tabelle in einer Anfrage oder auch in einer anderen Sicht verwendet werden. Eine Sicht ermöglicht es, einem Benutzer oder einer Benutzergruppe exakt die Daten zu präsentieren, die benötigt werden.

Die Sicht "CurrentProductList" enthält alle Produkte, die noch gepflegt werden, d.h. für die Discontinued=No gilt.

```
CREATE VIEW CurrentProductList AS
SELECT ProductID, ProductName
FROM Products
WHERE Discontinued=No
```

Anfrage an die Sicht

```
SELECT * FROM CurrentProductList
```

Eine weitere Sicht selektiert Produkte, deren Preis höher ist als der Durchschnittspreis.

```
CREATE VIEW ProductsAboveAveragePrice AS
SELECT ProductName, UnitPrice
FROM Products
WHERE UnitPrice>(SELECT AVG(UnitPrice) FROM Products)
```


Anfrage

```
SELECT * FROM ProductsAboveAveragePrice
```

Die Sicht CategorySalesFor1997 basiert auf der Sicht ProductSalesFor1997(CategoryName,Product,ProductSales,...).

```
CREATE VIEW CategorySalesFor1997 AS
SELECT DISTINCT CategoryName, SUM(ProductSales) AS CategorySales
FROM ProductSalesFor1997
GROUP BY CategoryName
```

Anfrage

```
SELECT * FROM CategorySalesFor1997
```

Anfrage mit einer Bedingung

```
SELECT * FROM CategorySalesFor1997
WHERE CategoryName='Beverages'
```

Zusammenfassung: SQL - DBMS - RDBMS

DBMS - Datenbankmanagementsystem

Ein Datenbankmanagementsystem (DBMS) ist ein Programm, das auf Daten in einer Datenbank zugreifen kann.

Ein DBMS setzt Benutzer in die Lage, Informationen aus einer Datenbank zu extrahieren, Informationen zu modifizieren oder zu speichern.

Unterschiedliche DBMSe stellen unterschiedliche Funktionen zum Anfragen, zur Berichterstattung und zur Modifikation zur Verfügung.

RDBMS - Relationales Datenbankmanagementsystem

Ein relationales Datenbankmanagementsystem (RDBMS) ist ein Datenbankmanagementsystem, in dem die Daten in Form von Tabellen (Relationen) organisiert werden.

RDBMSe wurden u.a. von IBM kurz nach 1970 entwickelt.

RDBMSe sind die Basis für SQL und für viele moderne Datenbanksysteme wie Oracle, SQL Server, IBM DB2, Sybase, MySQL, PostgreSQL und Microsoft Access.

Zentrale Begriffe im Bereich relationale Datenbanksysteme

- Relationales Datenbankschema: Sammlung von relationalen Schemata zur Darstellung eines Weltausschnitts
- Relationales Schema, Datenbanktabelle: Strukturbeschreibung mit Tabellenname, Spaltennamen und zugehörigen Datentypen
- Datenbankzustand, Zustand, Schemaausprägung, Ausprägung: Konkrete Daten zu einem Datenbankschema, die den Weltausschnitt (in der Regel) zu einem Zeitpunkt in Form von Tabellen bzw. Relationen wiedergeben
- Tabelle, Relation: Ausprägung eines relationalen Schemas
- Attribut, Spalte: Eigenschaft im beschriebenen Weltausschnitt
- Zeile, Tupel, Satz, Record: Eigenschaftsausprägungen im Datenbankzustand
- Anfrage, Query, Abfrage: Ermitteln von Informationen aus dem Datenbankzustand
- Unteranfrage, Subquery: Teil einer Anfrage, der wieder als Anfrage aufgefasst werden kann
- Integritätsbedingung, Constraint: Mechanismus zur Gewährleistung korrekter Datenbankzustände
- Schlüssel, Primärschlüssel, Alternativschlüsssel, Fremdschlüssel: Typische Integritätsbedingungen in einem relationalen Datenbankschema
- Sicht, View: Virtuelle, nicht explizit abgespeicherte Relation
- Index: Struktur zur effizienten Anfragebearbeitung
- Transaktion: Zusammenfassung von Datenbankänderungen zu einer Einheit, die von einem korrekten Datenbankzustand zu einem nächsten korrekten Datenbankzustand führt, wobei zwischenzeitlich die Integrität verletzt sein darf

---

SQL Quick Reference

SQL Quick Reference (W3Schools).

SQL Syntax

Statement	Syntax
AND / OR	SELECT column_name(s) FROM table_name WHERE Condition AND|OR Condition
ALTER TABLE (add column)	ALTER TABLE table_name ADD column_name datatype
ALTER TABLE (drop column)	ALTER TABLE table_name DROP COLUMN column_name
AS (Alias für eine Spalte)	SELECT column_name AS column_alias FROM table_name
AS (Alias für eine Tabelle)	SELECT column_name FROM table_name AS table_alias
BETWEEN	SELECT column_name(s) FROM table_name WHERE column_name BETWEEN value1 AND value2
CREATE DATABASE	CREATE DATABASE database_name
CREATE INDEX	CREATE INDEX index_name ON table_name (column_name)
CREATE TABLE	CREATE TABLE table_name (column_name1 data_type, column_name2 data_type, ...)
CREATE UNIQUE INDEX	CREATE UNIQUE INDEX index_name ON table_name (column_name)
CREATE VIEW	CREATE VIEW view_name AS SELECT column_name(s) FROM table_name WHERE Condition
DELETE FROM	DELETE FROM table_name (Löscht die gesamte Tabelle) *oder* DELETE FROM table_name WHERE condition
DROP DATABASE	DROP DATABASE database_name
DROP INDEX	DROP INDEX table_name.index_name
DROP TABLE	DROP TABLE table_name
GROUP BY	SELECT column_name1,SUM(column_name2) FROM table_name GROUP BY column_name1

HAVING	**SELECT column_name1,SUM(column_name2) FROM table_name GROUP BY column_name1 HAVING SUM(column_name2) Condition value**
IN	**SELECT column_name(s) FROM table_name WHERE column_name IN (value1,value2,..)**
INSERT INTO	**INSERT INTO table_name VALUES (value1, value2,....) *oder* INSERT INTO table_name (column_name1, column_name2,...) VALUES (value1, value2,....)**
LIKE	**SELECT column_name(s) FROM table_name WHERE column_name LIKE pattern**
ORDER BY	**SELECT column_name(s) FROM table_name ORDER BY column_name [ASC|DESC]**
SELECT	**SELECT column_name(s) FROM table_name**
SELECT *	**SELECT * FROM table_name**
SELECT DISTINCT	**SELECT DISTINCT column_name(s) FROM table_name**
SELECT INTO (um z.B. Kopien zu erstellen)	**SELECT * INTO new_table_name FROM original_table_name *oder* SELECT column_name(s) INTO new_table_name FROM original_table_name**
TRUNCATE TABLE (Löscht nur die Sätze in der Tabelle)	**TRUNCATE TABLE table_name**
UPDATE	**UPDATE table_name SET column_name=new_value [, column_name=new_value] WHERE column_name=some_value**
WHERE	**SELECT column_name(s) FROM table_name WHERE Condition**

Weitere Themen

- **SQL Syntax anhand PostgreSQL Reference**
- **Einschränkungen bei Sichten**
- **Rechte**
- **Drei-Ebenen-Architektur, DB-Administration, Funktionen eines DBMS**