SQL-Anweisungen

SELECT (SQL Data Query Language)	(siehe auch "Einfache Beispiele für SQL-Kommandos")
`SELECT * FROM meineTabelle;`	Alle Daten einer Tabelle lesen.
`SELECT * FROM "meine Tabelle";`	Normalerweise unterscheidet SQL nicht zwischen Groß-/Kleinschreibung. Wird der Tabellenname in Anführungszeichen gesetzt, muss Groß-/Kleinschreibung exakt stimmen und der Tabellenname immer genau so geschrieben werden. Einige Datenbanken akzeptieren dann auch Leerzeichen im Tabellennamen (was eigentlich nicht erlaubt ist).
`SELECT feldName1, feldName2 FROM meineTabelle;`	Bestimmte Felder (Spalten) einer Tabelle lesen.
`SELECT feldName1, feldName2 FROM meineTabelle ORDER BY feldName2, feldName1 DESC;`	Spalte(n) zur Sortierung vorgeben, entweder per Feldnamen oder auch per Spaltennummern. Bei Spaltennummern beachten: Die erste Spalte ist 1 (und nicht 0). Ohne `DESC` aufsteigend, mit absteigend.
`SELECT * FROM meineTabelle WHERE feldName1 = 'xy' AND feldName2 < 100 AND feldName3 BETWEEN 1 AND 10;`	Die Zeilen der Tabelle lesen, deren Elemente die Bedingung erfüllen. '=' testet auf Gleichheit, '<>' auf Ungleichheit und '<', '<=', '>' und '>=' vergleichen. Textstrings werden z.B. für Oracle, MySQL und MS Access mit einfachen Hochkommas, aber z.B. für InterBase mit doppelten Hochkommas eingeschlossen.
`SELECT * FROM meineTabelle WHERE UPPER(feldName1) = UPPER('xy');`	Vergleich mit Ignorierung von Groß-/Kleinschreibung. Kommandos sind unterschiedlich je nach Datenbank. Großschreibung wird z.B. bei Oracle mit UPPER() und bei MS-Access mit UCASE() erreicht.
`SELECT * FROM meineTabelle WHERE feldName1 LIKE 'B%';`	Die Zeilen der Tabelle lesen, deren Element in der Spalte feldName1 mit einem großen B beginnt (oder mit `'%abc%'` den Teilstring 'abc' enthält). '_' ist Platzhalter für genau einen Zeichen, '__' für zwei Zeichen und '%' für eins oder mehrere Zeichen.
`SELECT * FROM meineTabelle WHERE feldName1 IN( 11, 13, 17 );`	Selektiere Zeilen, wo feldName1 in angegebener Menge enthalten ist.
`SELECT * FROM meineTabelle1 WHERE feldName1 IN( SELECT feldName2 FROM meineTabelle2 );`	Wie vorher, aber angegebene Menge ist Resultat von weiterer Abfrage (mit einspaltigem Ergebnis).
`SELECT meineTabelle1.feldName3,`	Join zweier Tabellen. Leider ist die Syntax nicht bei allen Datenbanken

```
meineTabelle2.feldName4
FROM meineTabelle1,
meineTabelle2
WHERE
meineTabelle1.fremdSchlüsse
lFeld =
meineTabelle2.primärSchlüss
elFeld;
```

gleich. Die gezeigte Schreibweise gilt z.B. für Oracle, MySQL und MS Access.
Primärschlüsselspalte und Fremdschlüsselspalte können in der Datenbank entsprechend definiert werden.

```
SELECT Autor.Name,
Autor.Vorname, Buch.Titel,
Gebiet.Bez,
Verlag.Name_Kurz
FROM Autor, Buch, Gebiet,
Verlag
WHERE Buch.Autor_Nr =
Autor.Nr
AND Buch.Gebiet_Abk =
Gebiet.Abk
AND Buch.Verlag_Nr =
Verlag.Nr;
```

Join vierer Tabellen.
Bei Verknüpfung von n Tabellen sind n-1 Join-Kriterien erforderlich.

```
SELECT * FROM Kunde K
JOIN Bestellung B ON
K.kdkey=B.kdkey;
```

Join zweier Tabellen in einer für die Datenbank InterBase verständlichen Syntax.

```
SELECT
feldName1 "Nachname",
feldName2 "Vorname"
FROM meineTabelle;
```

Aliasnamen: Für Feldnamen andere Bezeichnungen vorgeben.

```
SELECT Nachname || ', ' ||
Vorname "Name"
FROM meineTabelle;
```

Konkatenation mit ||: Zwei Spalten werden zu einer Ausgabespalte (mit dem neuen Namen "Name") verbunden.

```
SELECT SUBSTR( Name, 1, 1 )
FROM meineTabelle;
```

Teilstring extrahieren. Parameter: String, Startposition, Länge.

```
SELECT DISTINCT feldName1
FROM meineTabelle;
```

DISTINCT bedeutet Zusammenfassung gleicher Elemente zu einer Zeile.

```
SELECT COUNT(*) "Anzahl"
FROM meineTabelle;
```

Eingebaute Aggregatfunktionen: COUNT() (Anzahl), MIN(), MAX(), AVG() (Durchschnitt), SUM().

```
SELECT ZahlungsEmpfaenger,
SUM(Betrag) FROM Rechnungen
GROUP BY
ZahlungsEmpfaenger;
```

GROUP BY reduziert die returnierten Reihen pro Group-Wert auf eine Reihe.
GROUP BY normalerweise zusammen mit Aggregatfunktionen (z.B. SUM, AVG ...).

```
SELECT TO_CHAR( Datum,
'YYYY' ) FROM meineTabelle;
```

Datentypkonvertierung: TO_CHAR() (String), TO_NUMBER() (Zahl), TO_DATE() (Datum).

```
SELECT * FROM meineTabelle
where date = TO_DATE(
'2002-01-23', 'YYYY-MM-
DD');
```

Datumsformatkonvertierung mit TO_DATE() (z.B. bei Oracle).

```
SELECT SYSDATE FROM DUAL;
```

SYSDATE ist das aktuelle System-Datum. DUAL ist

ein Dummy-Name als Platzhalter für eine Tabelle, wo eigentlich keine Tabelle benötigt wird. SYSDATE und DUAL werden nicht von allen Datenbanken unterstützt (aber z.B. von Oracle).

```
SELECT * FROM meineTabelle
WHERE Datum >= (SYSDATE -
28);
```

Die Zeilen der Tabelle lesen, deren Eintrag im Datumsfeld nicht älter als vier Wochen ist. Datums-Kommando ist unterschiedlich je nach Datenbank, z.B. SYSDATE bei Oracle und NOW() bei MS-Access.

```
SELECT 1 FROM DUAL WHERE
EXISTS
( SELECT 1 FROM
MeineTabelle WHERE ... );
```

EXISTS prüft Existenz.

```
SELECT * FROM meineTabelle
WHERE feldName1 IS NULL
AND feldName2 IS NOT NULL;
```

SQL returniert NULL, wenn ein Feld leer ist. Es gibt normalerweise keine Leerstrings. NULL kann nicht mit Vergleichsoperatoren geprüft werden, sondern mit IS NULL bzw. IS NOT NULL.

```
SELECT Name, NVL(
TO_CHAR(GebJahr), '?' )
FROM meineTabelle;
```

NVL() ersetzt NULL Values durch etwas anderes.

```
SELECT title, text
FROM books
WHERE CONTAINS( text,
'!door' ) > 0;
```

Ausrufezeichenoperator für phonetische Suche mit 'soundex' (nicht in allen Datenbanken implementiert, aber z.B. in Oracle).

DML (SQL Data Manipulation Language)

```
INSERT INTO tabelleAutor
( Nr, NachName, VorName,
GebJahr )
VALUES ( 1, 'Böll',
'Heinrich', 1917 );
```

Daten in bestehende Tabelle einfügen.
Zahlenwerte ohne Hochkommas und Datentextstrings mit einfachen Hochkommas angeben.
Soll der Wert eines Feldes nicht gesetzt werden, kann der entsprechende Feldname weggelassen werden oder alternativ als Datenelement NULL angegeben werden.

```
UPDATE tabelleAutor
SET Name = Otto, GebJahr =
1954, Beruf = NULL
WHERE Nr = 10;
```

Daten in Tabelle ändern.
Auf NULL setzen bedeutet Feld löschen.

```
DELETE FROM tabelleAutor
WHERE Datum < (SYSDATE -
3650);
```

Zeilen löschen (hier alle älter als 10 Jahre alten Einträge).

```
COMMIT;
```

Transaktion: Die seit dem vorherigen COMMIT-Kommando eingegebenen SQL-DML-Kommandos wirklich ausführen.

```
ROLLBACK;
```

Transaktion: Die seit dem vorherigen COMMIT-Kommando eingegebenen SQL-DML-Kommandos rückgängig machen.

```
LOCK TABLE meineTabelle
IN EXCLUSIVE MODE NOWAIT;
```

Locking einer ganzen Tabelle (bis zum nächsten COMMIT oder ROLLBACK).
(Locking einzelner Reihen geschieht automatisch bei Änderungen.)

```
SELECT * FROM meineTabelle
WHERE meinFeldname = 'xy'
FOR UPDATE OF meinFeldname;
```

Locking bestimmter per SELECT ... WHERE ... ausgewählter Datensätze (bis zum nächsten COMMIT oder ROLLBACK) schon beim Lesezugriff, damit zwischen Lesezugriff und späterer Änderung kein anderer Benutzer diese Datensätze ändern kann.

DDL (SQL Data Definition Language)

```
CREATE TABLE Autor (
Nr INT
CONSTRAINT Pk_Autor
PRIMARY KEY,
Name VARCHAR(80)
CONSTRAINT Nn_Autor_Name
NOT NULL,
VorName VARCHAR(80)
CONSTRAINT Nn_Autor_VorName
NOT NULL,
GebJahr INT,
Geschl CHAR(1)
CONSTRAINT Ch_Autor_Geschl
CHECK ( TYP IN ('m', 'w') )
,
UNIQUE( Nr ),
UNIQUE( Name, VorName ) );
```

Tabelle anlegen.
Namen/Bezeichner dürfen bis 30 Zeichen lang sein und keine Leerzeichen, Umlaute oder Sonderzeichen enthalten.
Die SQL-Datentypen sind in unterschiedlichen Datenbanken leider verschieden definiert. Siehe dazu auch unten die Tabelle 'SQL-Datentypen und -Funktionen in unterschiedlichen Datenbanken'.
`INT` und `FLOAT` kann in allen SQL-Datenbanken verwendet werden.
`NUMERIC(p,s)` (oder `DECIMAL(p,s)`, oder `NUMBER(p,s)`) speichert bei einigen Datenbanken Fließkommazahlen mit Angabe der maximalen Stellenanzahl und Nachkommastellenzahl.
`DATE` ist entweder nur Datum oder Kombination aus Datum und Uhrzeit.
`CHAR(n)` reserviert Speicherplatz in der festen angegebenen Länge (höchstens 255 Zeichen).
`VARCHAR(n)` definiert Strings variabler Länge. `n` gibt die maximale Länge an (je nach Datenbank höchstens 255 oder 2000 Zeichen).
Die Datentypen für lange Texte oder Binärdaten (z.B. Bilder) heißen bei verschiedenen Datenbanken unterschiedlich, z.B. `BLOB` (Binary Large Object), `LONGBLOB`, `LONG RAW`, `LONG`, `LONGCHAR`, `MEMO`. Die maximale Größe kann z.B. 64 KByte oder 4 GByte betragen.
`CONSTRAINT ... PRIMARY KEY` definiert Primärschlüsselspalten.
`CONSTRAINT ... REFERENCES` definiert Fremdschlüsselspalten.
`CONSTRAINT ... NOT NULL` erzwingt Eingabewerte.
`CONSTRAINT ... CHECK ...` ermöglicht zusätzliche Prüfungen.
`UNIQUE` mit einem Parameter stellt sicher, das es

	keine zwei gleichen Elemente in dieser Spalte gibt. Bei zwei Parametern gilt das Gleiche für Element-Kombinationen.
`ALTER TABLE meineTabelle MODIFY ( Nr NUMERIC(5) );`	Änderung an bestehender Tabellenstruktur.
`ALTER TABLE meineTabelle ADD ( Tel NUMERIC(20), Fax NUMERIC(20) );`	Spalten hinzufügen.
`DROP TABLE meineTabelle INCLUDING CONTENTS;`	Tabelle löschen.

`CREATE SEQUENCE Buch$Nr INCREMENT BY 1 MINVALUE 1;`
`INSERT INTO Buch ( Nr, Autor_Nr, Titel )`
`VALUES ( Buch$Nr.NEXTVAL, 10, 'meinBuchTitel' );`
	Fortlaufende Nummer vergeben (z.B. für Primärkey).

`ALTER SESSION SET nls_date_format = 'YYYY-MM-DD';`	Format der Datumsangabe für Oracle-Datenbank ändern.

DCL (SQL Data Control Language)

`GRANT SELECT, DELETE, UPDATE, REFERENCES(Nr) ON meineTabelle TO Mueller;`	Rechte vergeben.
`REVOKE DELETE ON meineTabelle FROM Mueller;`	Rechte entziehen.