Abfragen (Queries, Subqueries)

Grundstruktur einer SQL-Abfrage (reine Projektion)

```
SELECT [DISTINCT] {* | Spaltenname [[AS] "Aliasname" ] | Ausdruck} *
FROM Tabellenname;
```

Beispiele

1. Auswahl aller Spalten

   ```
   SELECT *
   FROM Tabellenname;
   ```

2. Auswahl bestimmter Spalten

   ```
   SELECT Spaltenname1, Spaltenname2
   FROM Tabellenname;
   ```

3. Auswahl bestimmter Spalten mit Definition von Aliasnamen

   ```
   SELECT Spaltenname1 as MeineSpalte1, Spaltenname2 as MeineSpalte2
   FROM Tabellenname;
   ```

4. Verwendung von Ausdrücken

   ```
   SELECT Spaltenname1 + 100 / 26 as MeineBerechnung1
   FROM Tabellenname;
   ```

   Hinweis: Spaltenname1 muss hierfür numerisch definiert sein.

5. Auswahl mit Unterdrückung von Doppelnennungen

   ```
   SELECT DISTINCT Spaltenname2
   FROM Tabellenname;
   ```

SQL-Abfragen mit Selektion

```
SELECT [DISTINCT] {* | Spaltenname [[AS] "Aliasname" ] | Ausdruck} *
FROM Tabellenname
WHERE Bedingungen;
```

```
Hinweis:
Eine Bedingung wird mit Hilfe von Vergleichsoperatoren definiert.
Die gängigsten Vergleichsoperatoren sind:
= > >= < <= <> IS NULL IS NOT NULL BETWEEN...AND... IN (Liste) und LIKE

Mehrere Bedingungen werden über die logischen Operatoren AND, OR und NOT
miteinander verknüpft.
```

Beispiele für die Bedingungen in der WHERE-Klausel

Hinweis: der Where-Klausel steht ein `SELECT … FROM …` vor.

1. Auswahl der Zeilen, wo der Spaltenname1 null ist:
   …
   WHERE Spaltenname1 is null;

2. Auswahl der Zeilen, wo die Spalteninhalte von Spaltenname1 und Spaltenname2 gleich sind:
   …
   WHERE Spaltenname1 = Spaltenname2;

3. Auswahl der Zeilen, wo die Spalteninhalte von Spaltenname1 und Spaltenname2 gleich sind und der Wert von Spaltenname3 nicht null sein darf:
   …
   WHERE Spaltenname1 = Spaltenname2
     AND Spaltenname3 is not null;

4. Auswahl der Zeilen, wo die Spalteninhalte von Spaltenname1 und Spaltenname2 gleich sind oder der Wert von Spaltenname3 null ist:
   …
   WHERE Spaltenname1 = Spaltenname2
     OR Spaltenname2 is null;

Verwendung von Sortierung

```
SELECT [DISTINCT] {* | Spaltenname [[AS] "Aliasname" ] | Ausdruck} *
FROM Tabellenname
[WHERE Bedingungen]
[ORDER BY {Spaltenname}* | Spaltenposition | Ausdruck] [ASC|DESC];

Hinweis:
ASC – aufsteigende Sortierung (DEFAULT)
DESC – absteigende Sortierung
```

ORDER BY steht, wenn es verwendet wird, IMMER zuletzt im SQL-Statement!
Die Spalte, nach der sortiert wird, muss nicht in der SELECT-Klausel vorkommen.

Join‘s

Equijoin

```
SELECT [DISTINCT] {* | TabellenAlias.Spaltenname | Ausdruck} *
FROM Tabelle1 TabellenAlias1, Tabelle2 TabellenAlias2
WHERE TabellenAlias1.Spalte1 = TabellenAlias2.Spalte1;

oder alternativ:

SELECT [DISTINCT] {* | TabellenAlias.Spaltenname | Ausdruck} *
FROM (Tabelle1 TabellenAlias1 [INNER] JOIN Tabelle2 TabellenAlias2
ON TabellenAlias1.Spalte1 = TabellenAlias2.Spalte1)
WHERE ( Bedingungen ) ;
```

Outerjoin

```
SELECT [DISTINCT] {* | TabellenAlias.Spaltenname | Ausdruck} *
FROM Tabelle1 TabellenAlias1, Tabelle2 TabellenAlias2
WHERE TabellenAlias1.Spalte1 [(+)] = TabellenAlias2.Spalte1 [(+)];

(+) steht bei der Tabellenspalte, wo die fehlende Information durch NULL
ergänzt werden soll.

oder alternativ:

SELECT [DISTINCT] {* | TabellenAlias.Spaltenname | Ausdruck} *
FROM (Tabelle1 TabellenAlias1 [LEFT|RIGHT|FULL|OUTER] JOIN Tabelle2
TabellenAlias2
ON TabellenAlias1.Spalte1 = TabellenAlias2.Spalte1)
WHERE ( Bedingungen ) ;
```

Gruppenfunktionen

```
SELECT Spaltenname1, Gruppenfunktion(Spaltenname2), . . .
FROM Tabellenname
[WHERE Bedingung(en)]
[GROUP BY Gruppenausdruck]
[HAVING Gruppenbedingung]
[ORDER BY {Spaltenname} | Ausdruck | Aliasname} [ ASC | DESC ] ];
```

Hinweis:
Gruppenfunktionen können nicht in der WHERE-Klausen verwendet werden, hierfür gibt es die HAVING-Klausel, sie verhält sich wie die WHERE-Klausel, nur auf Gruppen bezogen.

Gruppenfunktionsübersicht

Gruppenfunktion	*Bedeutung*
`AVG (Gruppenelement)`	Ermittelt den Durchschnittswert
`SUM (Gruppenelement)`	Ermittelt die Summe des Gruppenelements
`MIN (Gruppenelement)`	Ermittelt den kleinsten Wert d. Gruppenelements
`MAX (Gruppenelement)`	Ermittelt den größten Wert d. Gruppenelements
`STDDEV (Gruppenelement)`	Ermittelt den Wert der Standardabweichung
`VARIANCE (Gruppenelement)`	Ermittelt den Wert f. d. Varianz
`COUNT (Gruppenelement)`	Zählt die Werte d. Gruppenelements
`COUNT DISTINCT (Gruppenelement)`	Ermittelt die Anzahl unterschiedlicher Werte d. Gruppenelements
`COUNT (*)`	Ermittelt die Anzahl Zeilen in einer Tabelle

```
Gruppenelement := Spalte | Ausdruck
```

Beispiele für die Verwendung von Gruppenfunktionen

Zählt die Anzahl Zeilen in Tabelle Tabellenname:

```
SELECT COUNT(*)
FROM Tabellenname;
```

Zählt die Anzahl Zeilen für jeden Wert in Spaltenname1 in Tabelle `Tabellenname`:

```
SELECT Spaltenname1, COUNT(*)
FROM Tabellenname
GROUP BY Spaltenname1;
```

Bildet für jedes Wertpaar Spaltenname1, Spaltenname2 die Summe von Spaltenname3

```
SELECT Spaltenname1, Spaltenname2, SUM(Spaltenname3)
FROM Tabellenname
GROUP BY Spaltenname1, Spaltenname2;
```

Bildet die Summe von Spaltenname1, den Durchschnittswert für Spaltenname1 und den größten Wert von Spaltenname2 aus Tabelle Tabellenname:

```
SELECT SUM(Spaltenname1), AVG(Spaltenname1), MAX(Spaltenname2)
FROM Tabellenname;
```

Für jeden Wert von Spaltenname1 wird die Anzahl Zeilen ermittelt. Allerdings nur dann, wenn der maximale Wert von Spaltenname1 grösser als 100 ist. Die Ausgabe wird nach Spaltenname4 sortiert:

```
SELECT Spaltenname1, COUNT(*)
FROM Tabellenname1
WHERE Spaltenname3 is not null
GROUP BY Spaltenname1, Spaltenname2
HAVING MAX(Spaltenname1) > 100
ORDER BY Spaltenname4;
```

Subqueries / Unterabfragen

Subqueries können in folgenden SQL-Klauseln verwendet werden:
SELECT, FROM, WHERE, HAVING implizit auch in der GROUP BY und ORDER BY-Klausel, da sich dort auf einen Wert bezogen werden kann.

```
SELECT {Spaltenname | Ausdruck | Subquery} *
FROM Tabellenname | Subquery
[WHERE Bedingung(en)]
[GROUP BY Gruppenausdruck]
[HAVING Gruppenbedingung]
[ORDER BY {Spaltenname} | Ausdruck | Aliasname} [ ASC | DESC ] ];
```

WHERE – Bedingungen

Subqueries, die einen Wert zurückliefern

```
SELECT {Spaltenname | Ausdruck | Subquery} *
FROM Tabellenname
WHERE Spaltenname Operation (Select-Statement)
[ AND . . . ];
```

Subqueries, die mehrere Werte zurückliefern

```
Operatoren: IN; ANY; ALL; EXISTS

SELECT {Spaltenname | Ausdruck | Subquery} *
FROM Tabellenname
WHERE Spaltenname {{=,>,...} [ANY|ALL] | IN }
      (SELECT Spaltenname
      FROM Tabellenname
      [ WHERE Bedingung(en) ] );


SELECT {Spaltenname | Ausdruck | Subquery} *
FROM Tabellenname
WHERE (Spaltenname1, Spaltenname2,... ) [= | IN]
      (SELECT Spaltenname1, Spaltenname2, ...
      FROM Tabellenname
      [ WHERE Bedingung(en) ] );


SELECT {Spaltenname | Ausdruck | Subquery} *
FROM Tabellenname
WHERE EXISTS
      (SELECT . . .
      FROM Tabellenname
      [ WHERE Bedingung(en) ] );
```

FROM-Klausel (Inline View)

```
SELECT {Spaltenname | Ausdruck | Subquery} *
FROM Tabellenname1 Aliasname1,
(SELECT {Spaltenname | Ausdruck | Subquery} *
FROM Tabellenname
GROUP BY ... ) Aliasname2
WHERE Aliasname1.Spaltenname1 = Aliasname2.Spaltenname2
[ AND ... ];
```

Data Manipulation Language DML

DML-Kommandos verändern die Inhalte in Tabellen. Sie erzeugen, ändern und löschen Datenzeilen.

Insert - Einfügen neuer Datensätze

```
INSERT INTO Tabellenname [(Spalten-Liste)]
VALUES ( Werte-Liste ) ;
```

Datensätze einer Tabelle in eine andere Tabelle kopieren.

```
INSERT INTO Tabellenname1 [(Spalten-Liste)]
(SELECT {Spaltenname | Ausdruck | Subquery} *
FROM Tabellenname2
[WHERE Bedingung(en)] );
```

Update - Ändern von Datensätzen

```
UPDATE Tabellenname
SET Spaltenname1 = Wert1 , [Spaltenname2 = Wert2 , . . . ]
[WHERE Bedingung(en)];
```

Ändern von Datensätzen basierend auf anderen Tabellen

```
UPDATE Tabellenname1
SET
Spaltenname1 =
(SELECT {Spaltenname | Ausdruck | Subquery} *
 FROM Tabellenname2
[WHERE Bedingung(en)]),
Spaltenname2 =
(SELECT {Spaltenname | Ausdruck | Subquery} *
FROM Tabellenname 3
[WHERE Bedingung(en)]);
```

Delete - Löschen von Datensätzen

```
DELETE [FROM] Tabellenname
[WHERE Bedingung(en)] ;

bzw.

DELETE [FROM] Tabellenname
WHERE Spaltenname=
(SELECT {Spaltenname | Ausdruck | Subquery} *
[WHERE Bedingung(en)] );
```

Truncate Table – Tabellen löschen – mit Speicherfreigabe

```
TRUNCATE TABLE Tabellenname;
```

Data Definition Language DDL

DDL-Kommandos erzeugen, ändern und löschen Datenbankobjekte. Weiterhin dienen sie dem Verwalten von Rechten.

Create Table – Tabellen erzeugen

```
CREATE TABLE Tabellenname
(Spaltenname1 Datentyp [DEFAULT Ausdruck]
[, Spaltenname2 Datentyp [DEFAULT Ausdruck], . . . ]);
```

Alter Table – Tabellen verändern

```
ALTER TABLE Tabellenname {ADD | MODIFY}
(Spaltenname1 Datentyp1 [DEFAULT Ausdruck]
[, Spaltenname2 Datentyp2 [DEFAULT Ausdruck], . . . ]);

ALTER TABLE Tabellenname
DROP COLUMN Spaltenname1;
```

Drop Table – Tabellen löschen

```
DROP TABLE Tabellenname;
```

Transaktionssteuerung

Anweisung	*Bedeutung*
`COMMIT`	Beendet die akt. Transaktionund schreibt Datenänderungen fest.
`SAVEPOINT name`	Markiert einen Savepoint innerhalt einer Transaktion
`ROLLBACK`	Beendet die akt. Transaktion und verwirft alle durchgeführten Änderungen
`ROLLBACK TO SAVEPOINT name`	Beendet die Transaktion und verwirft alle durchgeführten Änderungen bis zum Savepoint

Beispiel – das Insert wird Rückgängig gemacht

```
UPDATE …;

SAVEPOINT update_done;              -- Savepoint created

INSERT …;

ROLLBACK TO update_done;            -- Rollback complete
```