



SQL

Structured Query Language

Teil3:

Data Query Language



Die Datenbankabfragesprache SQL

1. Data Query Language

Die Data Query Language (DQL) ist eine Teilsprache von SQL und umfasst alle Anweisungen zum Abfragen von Daten bzw. Informationen. Die DQL stellt dabei Sprachelemente zum Lesen von in der Datenbasis gespeicherten Daten zur Verfügung. Einziger Befehl dieses Sprachbestandteils von SQL ist die SELECT-Anweisung.

Tabelle Lehrer

Die Tabelle Lehrer des Relationalen Datenmodells aus dem Skript DDL / DML wurde um folgende Attribute erweitert. Alle SQL-Abfrage-Beispiele beziehen sich auf diese Datenschema der Tabelle Lehrer.

<u>(PS) l_id</u> [int]
pnr [char(6)]
name [varchar(100)]
vorname [varchar(100)]
gebdat [date]
gebort [varchar(100)]
gehalt [decimal(8,2)]
(FS) abteilungs_id [varchar(5)]

Beispieldatensatz

<u>l_id</u>	pnr	name	vorname	gebdat	gebort	gehalt	abt_id
15	002560	Muster	Max	1976-01-25	Wien	54598,26	IT

2. SELECT-Statement

Die **Grundform** eines SELECT-Statements in SQL besteht aus folgenden Komponenten:

- der **SELECT**-Klausel => projiziert die hier angegebenen Spalten
- der **FROM**-Klausel => definiert die Ausgangstabellen
- der **WHERE**-Klausel => selektiert die Reihen, die der Bedingung genügen
- der **GROUP BY**-Klausel => gruppiert Reihen auf Basis gleicher Werte in Spalten
- der **HAVING**-Klausel => selektiert Gruppen, die der Bedingung genügen
- der **ORDER BY**-Klausel => sortiert Reihen auf der Basis von Spalten

Grundsätzlich sind **nur** die Klauseln **SELECT** und **FROM** **erforderlich**. Die restlichen Komponenten sind optional. Jedoch ist eine **Einhaltung der Reihenfolge** zwingend notwendig. Das bedeutet, dass eine GROUP BY Komponente nicht vor einer WHERE oder vor einer FROM Klausel stehen darf. Außerdem kann eine **HAVING-Komponente** **nur** dann verwendet werden, **wenn eine GROUP BY** Komponente **verwendet** wird.

Das komplette SELECT-Statement wird (wie alle SQL-Statements) mit einem Semikolon (;) abgeschlossen.

SQL-Klausel	Bedeutung	Angabe notwendig
SELECT	Auswahl der gewünschten Attribute	ja
FROM	Angabe der Tabelle(n), die die gewünschte Information enthalten	ja
WHERE	Angabe von Bedingungen für die gesuchte Information	nein
GROUP BY (HAVING)	Gruppieren der Daten nach bestimmten Kategorien	nein
ORDER BY	Sortierung des Ergebnisses	nein

In SQL stehen auch die **Aggregationsfunktionen** COUNT, AVG, SUM, MIN, MAX zur Verfügung.

Syntax einer SQL-Abfrage

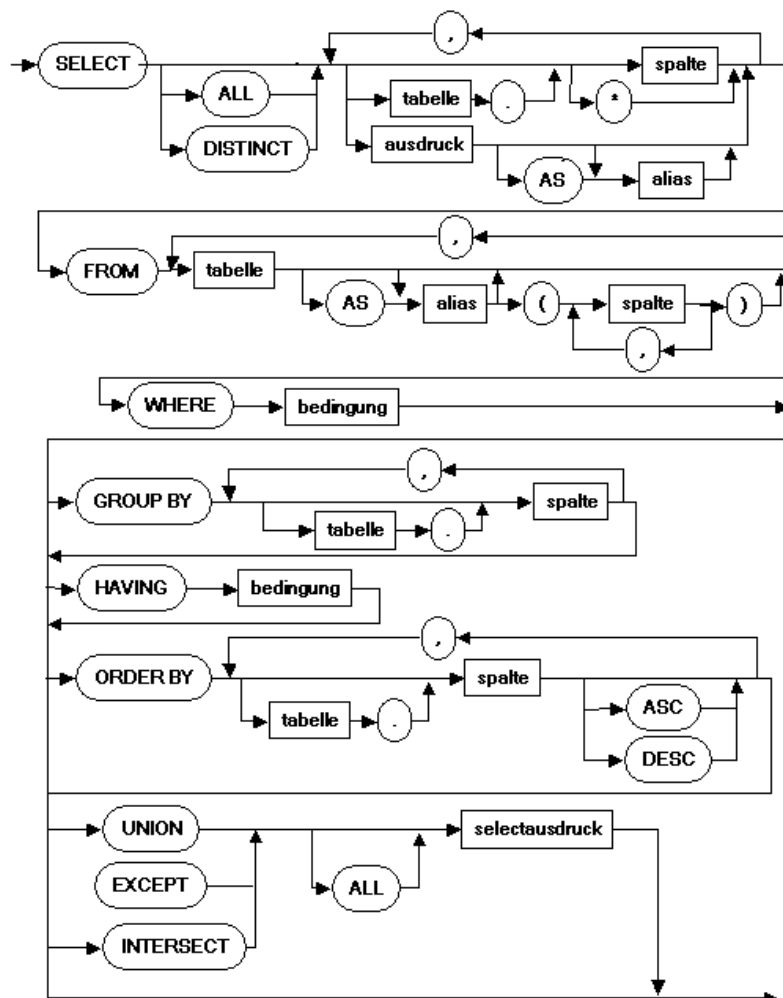
```

SELECT [DISTINCT] * | Datenfeld [, Datenfeld AS Alias]
FROM Tabellennamen
[WHERE Bedingung]
[GROUP BY Datenfeld [, Datenfeld] [HAVING Bedingung] ]
[ORDER BY Datenfeld [, Datenfeld] [ASC|DESC]]
[LIMIT [Start, ] Anzahl];

```

- Die Anweisung wird mit dem Schlüsselwort SELECT eingeleitet.
- Um doppelte (identische) Datensätze zu vermeiden, können Sie nach dem Schlüsselwort SELECT die Angabe DISTINCT hinzufügen.
- Danach folgt die Angabe der Namen der gewünschten Datenfelder. Die Datenfeldnamen müssen den bei der Tabellendefinition angegebenen Namen entsprechen. Die Feldnamen werden durch Kommata getrennt. Das Zeichen * steht für alle Datenfelder der Tabelle.
- Es kann zusätzlich mit dem Schlüsselwort AS für jedes Datenfeld der Abfrage ein Ersatznamen (Alias) definiert werden.
- Nach dem Schlüsselwort FROM folgt die Angabe der gewünschten Tabelle.
- Alle folgenden Angaben sind optional und können entfallen.

Syntaxdiagramm



3. Zeilen auswählen – Selektion mit der WHERE-Klausel

Der optionale **WHERE-Teil** (Selektion) in einem SELECT-Statement gibt **Bedingungen** an, mit denen die Menge der angezeigten **Datensätze** weiter **eingeschränkt** werden kann. Auf Basis der **Projektion** werden nun die gewünschten **Zeilen** (rows) ausgewählt und **angezeigt**. **Bedingungen** lassen sich mit den Operatoren **AND, OR und NOT** **verknüpfen**. Zur Einschränkung im Selektionsteil können folgende **Vergleichsoperatoren** bzw. Verknüpfungen verwendet werden:

Einfache Vergleichsoperatoren

=	gleich	>	größer	<	kleiner
<>	ungleich	>=	größer gleich	<=	kleiner gleich

Spezielle Vergleichsoperatoren

LIKE	Suche nach einem Muster mit Verwendung von Platzhaltern: '_' steht für genau ein beliebiges Zeichen '%' steht für eine beliebige Anzahl von beliebigen Zeichen
BETWEEN ... AND ...	Wertebereich mit definierter Unter- und Obergrenze Die angegebenen Grenzwerte in der Auswahl miteingeschlossen !
IN	Liste von Werten durch Kommata getrennt

Syntax einer Selektion mit der WHERE-Klausel

```

SELECT <Spalten>
FROM <Tabellenname>
WHERE <Bedingung>;

```

Beispiele auf Basis der Tabelle *Lehrer*

Selektion anhand einer bestimmten Lehrer-ID:

```
select name, vorname
from lehrer
where L_ID = 'WaFl';
```

Selektion des Geburtsdatums nach dem 01.01.1960:

```
select name as Nachname, vorname as Rufname
from lehrer
where geb_dat > '1960-01-01';
```

Selektion des Geburtsdatums und des Anfangsbuchstaben W vom Nachnamen:

```
select name, vorname
from lehrer
where geb_dat > '1960-01-01' and name like 'W%';
```

Selektion des Geburtsdatums in einem bestimmten Zeitraum:

```
select name, vorname
from lehrer
where geb_dat between '1970-01-01' and '1980-12-31';
```

Selektion über itbspa-Email-Adresse oder den Geburtsorten Passau, München und Wien:

```
select name, vorname
from lehrer
where email like '%itbspa%' or geb_ort in ('Passau', 'München', 'Wien');
```

4. Sortierung mit der ORDER BY-Klausel

Mit der ORDER BY-Klausel kann die Ergebnistabelle nach ein oder mehreren Kriterien sortiert werden. Durch Angabe von ASC wird aufsteigend, bei DESC absteigend sortiert. Fehlt die Angabe, wird immer aufsteigend sortiert.

Syntax

```
SELECT <Spalten>
FROM <Tabellenname>
[WHERE <Bedingung>]
ORDER BY <Spalte> [ASC | DESC];
```

Beispiel: Absteigende Sortierung nach dem Gehalt aller Lehrer der IT-Abteilung

```
select name, vorname, gehalt
from Lehrer
where abteilungs_id = 'IT'
order by gehalt desc;
```

5. Verdichtung mit der GROUP BY-Klausel

Mit der GROUP BY-Klausel werden bestimmte Sätze zu einem Datensatz in der Ergebnistabelle zusammengefasst. **Alle Datensätze, die die gleichen Werte**, in den hinter GROUP BY angegebenen Spalten besitzen, werden zu einem Datensatz **zusammengefasst**.

Syntax

```
SELECT <Spalten>
FROM <Tabellenname>
[WHERE <Bedingung>]
[GROUP BY <Gruppierungsspalten>]
[ORDER BY <Spalte> [ASC | DESC]];
```

Beispiel: Absteigende Sortierung der Geburtsorte aller Lehrer

```
select gebort as Geburtsorte_Lehrer
from Lehrer
group by gebort desc;
```

6. Aggregatsfunktionen

Im Zusammenhang mit Gruppierung spielen Aggregationsfunktion eine wichtige Rolle. Sie werden auf allen Datensätzen, die zu einer Gruppierung gehören, ausgeführt. Im Allgemeinen liefern Aggregatsfunktionen immer nur einen Wert zurück! Im Folgenden befindet sich eine Auswahl von wichtigen Aggregatsfunktionen.

COUNT	Anzahl von Werten
SUM	Summe von Werten
AVG	arithmetischer Mittelwert von Werten (SUM / COUNT)
MIN	kleinster Wert
MAX	größter Wert

Beispiele von Aggregatsfunktionen auf Basis der Tabelle Lehrer

Anzahl aller Lehrer:

```
select count(l_id) as Anzahl1_Lehrer
from Lehrer;
```

Anzahl der Lehrer pro Abteilung:

```
select count(l_id) as Anzahl1_Lehrer
from Lehrer
group by abteilungs_id;
```


Gesamtgehalt der Lehrer in der IT-Abteilung:

```
select sum(gehalt) as Gehalt_Lehrer_IT_Abteilung
from Lehrer
where abteilungs_id = 'IT';
```

Durchschnittsgehalt der Lehrer pro Abteilung:

```
select avg(gehalt) as Durchschnittsgehalt_Lehrer_pro_Abteilung
from Lehrer
group by abteilungs_id;
```

Geringstes Gehalt in der IT-Abteilung:

```
select min(gehalt)
from Lehrer
where abteilungs_id = 'IT';
```

7. HAVING-Klausel

Durch die WHERE-Klausel können nur Bedingungen zu Werten formuliert werden, die bereits in den Datensätzen vorhanden sind. Mit der HAVING-Klausel können allerdings auch Bedingungen mit Werten formuliert werden, die erst durch eine Aggregatfunktion berechnet werden. Die Having-Klausel ist daher nur in Verbindung mit GROUP BY zu verwenden!

Syntax

```
SELECT <Spalten>
FROM <Tabellenname>
[WHERE <Bedingung>]
[GROUP BY <Gruppierungsspalten>]
[HAVING <Bedingung2>];
```

Beispiele auf Basis der Tabelle *Lehrer*

Ausgabe des Maximalgehaltes pro Abteilung, welches aber über 4500 liegt:

```
select max(gehalt) as Maximalgehalt_pro_Abteilung, abteilungs_id
from Lehrer
group by abteilungs_id;
having max(gehalt) > 4500;
```

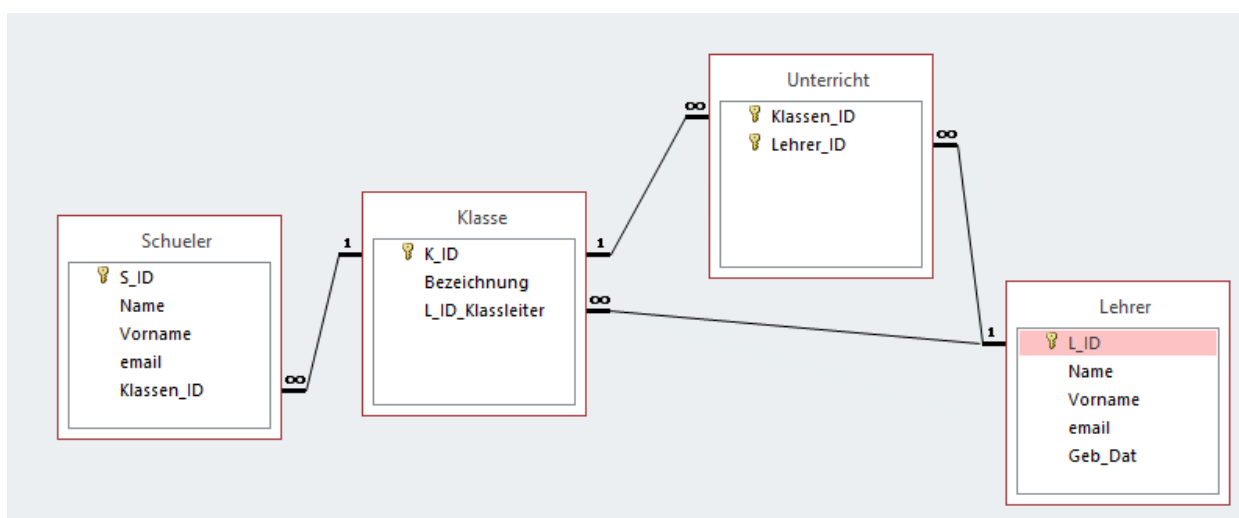
8. JOINS - Verknüpfungen

Joins dienen zur Verknüpfung mehrerer Tabellen. Verknüpfungskriterien können über WHERE-Klausel realisiert werden. Dies hat den Nachteil, dass sie nicht unbedingt als Joins erkennbar sind, da ja auch Filterbedingungen in der WHERE-Klausel auftauchen können.

Spezielle Join-Konstrukte sind auch nicht über die WHERE-Klausel abzubilden. Deshalb behandeln wir jetzt explizite Formulierungsmöglichkeiten für Joins. Damit kann auch die Performanz bei Abfragen optimiert werden.

Relationales Datenmodell der Schulverwaltung

Für die folgenden Beispiele ist dieses relationale Datenmodell die Grundlage.



8.1 CROSS JOIN

Der CROSS JOIN zweier Tabellen bildet das kartesische Produkt der Datensätze der beiden Tabellen. Dabei wird jeder Datensatz der ersten Tabelle mit jedem anderen der zweiten Tabelle verknüpft. D.h. die Ergebnistabelle besteht aus **Anzahl Datensätze Tabelle A * Anzahl Datensätze Tabelle B**. Wenn die beiden Tabellen gleichnamige Attribute haben, werden sie durch das Voranstellen des Tabellennamens ergänzt.

Diese Ergebnistabelle macht natürlich oftmals wenig Sinn. Daher müssen Einschränkungen über die WHERE-Klausel vorgenommen werden.

Syntax

```

SELECT <Spalten>
FROM <Tabellenname A>, <Tabellenname B>

```

Beispiel: Kartesisches Produkt aus Lehrer und Klasse

```
select *
from Lehrer, Klasse
```

8.2 INNER JOIN

Aus dem kartesischen Produkt von zwei Tabellen werden Datensätze selektiert, die eine spezifische Join-Bedingung erfüllen (hier: `Tabelle1.PrID = Tabelle2.PrID`). Wie der Name schon sagt, handelt es sich hierbei um den inneren Verbund zweier Tabellen. Das Ergebnis ist dabei die Kombination der Datensätze der beteiligten Tabellen, die die Verbundbedingung erfüllen. Das Schlüsselwort `INNER` kann auch weggelassen werden.

[illegible]

Syntax

SELCET <Spalten>

FROM **<Tabellenname1>** **INNER JOIN** **<Tabellenname2>** **ON** **<Bedingung>**

Beispiel: JOINS mit der Tabelle Lehrer und Klasse

INNER JOIN: Alle Informationen über Lehrer, die Klassenleiter sind

```
select *
from Lehrer as l inner join Klasse as k on l.l_id = k.l_id klassenleiter;
```

Zum Vergleich: CROSS-JOIN

```
select *
from Lehrer as l, Klasse as k
where l.l_id = k.l_id_klassenleiter;
```

8.3 LEFT-JOIN

Dies ist eine spezielle Art des INNER-Joins. Der Left-Join bzw. Left-OUTER-Join liefert zusätzlich zu den betrachteten INNER-Join die Zeilen der linken Tabelle, die nicht mit einer Zeile der rechten Tabelle verknüpft werden können, d.h. die keinen „Partner“ in der rechten Tabelle haben. Die fehlenden Daten der rechten Partnertabelle werden mit NULL-Werten aufgefüllt.

[illegible]

Beispiel: JOINS mit der Tabelle Lehrer und Klasse

LEFT JOIN: Alle Informationen über Lehrer und Klasse, die Klassenleiter sind oder nicht

```
select *
from Lehrer as l left join Klasse as k on l.l_id = k.l_id klassenleiter;
```

In der Ergebnismenge werden die Attribute von Klasse bei den Lehrern, die keine Klassenleiter sind, mit NULL-Werten befüllt!

8.4 RIGHT-JOIN

Der RIGHT-Join ist das Gegenstück des LEFT-Join. Dieser liefert zusätzlich zu den betrachteten INNER-Join die Zeilen der rechten Tabelle, die nicht mit einer Zeile der linken Tabelle verknüpft werden können, d.h. die keinen „Partner“ in der linken Tabelle haben. Die fehlenden Daten der linken Partnertabelle werden mit NULL-Werten aufgefüllt.

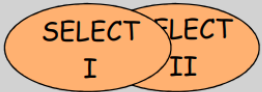
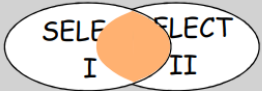

RIGHT JOIN: Alle Informationen über Lehrer und Klasse, die Klassenleiter sind oder nicht

```
select *  
from Klasse as k left join Lehrer as l on l.l_id = k.l_id_klassenleiter;
```

Identisches Ergebnis wie beim LEFT-Join, da die Tabelle Lehrer nun auf der rechten Seite der JOIN-Anweisung steht! D.h. ich kann jeden LEFT-Join auch in einen RIGHT-Join umwandeln. Die Umkehrrichtung (RIGHT in LEFT umwandeln) ist ebenfalls gegeben.

9. Mengenoperatoren

Mengenoperatoren verbinden jeweils zwei selektierte Datenmengen. Die folgenden drei Mengenoperatoren stehen zur Verfügung:

UNION	Vereinigung	
INTERSECT	Schnittmenge	
EXCEPT	Menge mit Datensätzen des ersten select, die nicht in zweitem select vorkommen	

Bezogen auf die „Ergebnis-Mengen“ der beiden SELECT-Statements funktioniert

union all

wie **union**, allerdings ohne Entfernung mehrfach vorkommender Datensätze

intersect all

wie **intersect**, allerdings erscheinen mehrfach vorkommende Datensätze in gleicher Zahl wie in dem **select**, in welchem sie weniger häufig vertreten sind

except all

wie **except**, allerdings kommen Datensätze in der Anzahl vor, die sich aus der Differenz der Anzahl ihres Auftretens in den beiden **selects** ergibt (sofern der Wert positiv ist)

9.1 UNION – Vereinigung

Durch die UNION-Klausel können Datensätze aus zwei Tabellen zusammengeführt werden. Dies ist möglich mit Tabellen gleicher als auch unterschiedlicher Datenstruktur! UNION entfernt im Gegensatz zu UNION ALL alle doppelten Zeilen.

Die Anweisung UNION ALL vereinigt die Ergebnismengen zweier Abfragen, wobei alle Werte, also auch mehrfach vorkommende Datensätze erhalten bleiben.

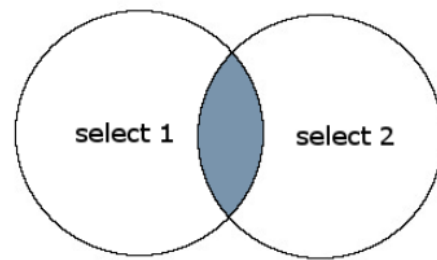
Versucht man eine Abfrage mit UNION-Klausel über zwei Tabellen mit unterschiedlicher Datenstruktur (z.B. unterschiedliche Zahl von Attributen) auszuführen kommt es zu einer Fehlermeldung. In diesem Fall hat man die Möglichkeit durch Auswahl einer gleichen Anzahl von Attributen die Abfrage auszuführen!

Syntax

```
SELCET <Auswahl>
FROM <Tabellenname1>
UNION [ALL]
SELCET <Auswahl>
FROM <Tabellenname2>;
```

9.2 INTERSECT – Schnittmenge

Mit dem INTERSECT-Operator von SQL Server (Transact-SQL) werden die Datensätze zurückgegeben, die zwei SELECT-Anweisungen oder -Datensätze gemeinsam haben. Wenn ein Datensatz in einer Abfrage und nicht in der anderen vorhanden ist, wird er in den INTERSECT-Ergebnissen nicht berücksichtigt. Es ist der Schnittpunkt der beiden SELECT-Anweisungen.

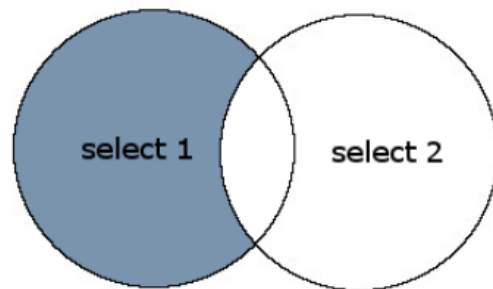


Syntax

```
SELCET <Auswahl>  
FROM <Tabellenname1>  
INTERSECT [ALL]  
SELCET <Auswahl>  
FROM <Tabellenname2>;
```

9.3 EXCEPT

Der SQL EXCEPT Operator wird verwendet, um alle Zeilen in der ersten SELECT-Anweisung zurückzugeben, die von der zweiten SELECT-Anweisung nicht zurückgegeben werden. Jede SELECT-Anweisung definiert eine Datenmenge. Der EXCEPT Operator ruft alle Datensätze aus dem ersten Datensatz ab und entfernt dann alle Datensätze aus dem zweiten Datensatz aus den Ergebnissen.



Syntax

```
SELCET <Auswahl>  
FROM <Tabellenname1>  
EXCEPT [ALL]  
SELCET <Auswahl>  
FROM <Tabellenname2>;
```

10. Nachschlagemöglichkeiten

Weitere Nachschlagemöglichkeiten finden Sie unter folgendem Link:

<https://www.w3schools.com/sql/default.asp>

Raum für Notizen:

This image shows a blank sheet of white paper with horizontal ruling lines. The lines are evenly spaced and extend across the width of the page. There are no margins, text, or other markings on the paper.