SQL, DQL, Daten abfragen und auswerten

Das **SELECT**-Statement ermöglicht es, Daten aus einer Datenbank abzurufen und auszuwerten. Dazu bietet das **SELECT**-Statement sehr viele Möglichkeiten. Im Folgenden werden häufig gebrauchte Befehle und Schlüsselwörter erklärt.

> **Vergessen Sie nicht:** In jedem DBS gibt es zusätzliche spezifische Erweiterungen, die Sie in der jeweiligen Dokumentation nachlesen können.

Die Anatomie des SELECT-Statements

```
SELECT [ALL|DISTINCT ]
    { col_name [as alias]|expression [as alias]
     [,col_name[as alias]|expression [as alias]]}
[FROM table_references ]
[WHERE where_condition]
[GROUP BY {col_name|expression|position}]
[HAVING where_condition]
[ORDER BY {col_name|expression|position} [ASC|DESC][,...]];
```

Der SELECT - Abschnitt

Dieser Abschnitt enthält entweder eine Spalte aus einer Tabelle zur reinen Anzeige oder einen Ausdruck, der Berechnungen mit Operatoren (s.u.) und Funktionen enthalten kann. Bei Berechnungen werden häufig mathematische Operatoren verwendet.

ALL

Das Schlüsselwort `ALL` gibt an, dass alle Datensätze angezeigt werden. Wird es weggelassen, werden standardmäßig alle Datensätze angezeigt.

DISTINCT

Verwirft alle Datensätze , die in der Ergebnismenge doppelt sind.

Der FROM – Abschnitt

In diesem Abschnitt wird die Tabelle oder werden die Tabellen aufgelistet, die die Spalten und damit auch die Daten enthalten, die ausgewertet werden sollen. Zusätzlich wird angegeben, wie das DBMS die Daten aus den verwendeten Tabellen verknüpfen soll. Das bedeutet, vorhandene PK- und FK-Beziehungen werden nicht automatisch verwendet. Darüber hinaus besteht die Möglichkeit, Daten auf jede gewünschte Weise zu verknüpfen – wenn das nötig und sinnvoll ist.

Der WHERE – Abschnitt

In diesem Abschnitt werden die Datensätze gefiltert. Das geschieht durch die Angabe von Suchkriterien. Hier können Spalten und Berechnungen in Verbindung mit Vergleichs-Operatoren und Suchkriterien verwendet werden.

Der GROUP BY – Abschnitt

In diesem Abschnitt werden Spalten angegeben, um aus Datensätzen Gruppen zu bilden. D.h. gleiche Datenkombinationen in den Gruppierungs-Spalten werden zu einem Datensatz zusammengefasst. Für diese Gruppen können mittels Aggregat-Funktionen statistische Werte berechnet werden: z.B. `MIN(.)`, `MAX(.)`, `SUM(.)`, `AVG(.)`, `COUNT(.)`. Diese Berechnungen werden im **SELECT**-Abschnitt festgelegt. Detaillierte Information über Gruppierungs-Abfragen finden Sie im Informationsblatt: *SQL, DQL, Gruppierung von Daten.*

Der `HAVING` – Abschnitt

Dieser Abschnitt dient dazu, berechnete Aggregate nach erfolgter Gruppierung und Berechnung über Kriterien zu filtern. D. h. in diesem Abschnitt werden Aggregat-Funktionen mit Vergleichs-Operatoren und Suchkriterien verwendet. Siehe: *SQL, DQL, Gruppierung von Daten.*

Wichtig: Es ist nicht möglich, berechnete Aggregate im WHERE-Abschnitt zu filtern!

Der `ORDER BY` – Abschnitt

In diesem Abschnitt wird angegeben, wie die Datensätze im Ergebnis sortiert werden sollen. Dazu werden die Spalten und berechnete Ausdrücke angegeben, die für die Sortierung berücksichtigt werden sollen. Hinter jeder Spalte oder jedem Ausdruck wird die Sortierreihenfolge angegeben.

ASC (engl. ascending)

ASC gibt an, dass die Werte dieser Spalte aufsteigend sortiert werden – also von kleinen hin zu großen Werten. ASC ist der Standardwert, wenn er hinter einer Spalte bzw. Einer Berechnung fehlt.

DESC (engl. descending)

DESC gibt an, dass die Werte dieser Spalte absteigend sortiert werden – also von großen hin zu kleinen Werten.

Liste wichtiger Operatoren

Operator	Beschreibung
Mathematische Operatoren	
`+`	Addition
`-`	Subtraktion
`*`	Multiplikation
`/`	Division
`%`	Modulo-Operator. Ermittlung des Rests einer Ganzzahldivision.
Vergleichs-Operatoren	
`=`	GLEICH
`<`	KLEINER als
`<=`	KLEINER oder GLEICH
`>`	GRÖSSER
`>=`	GRÖSSER oder GLEICH
`<>`	UNGLEICH
Logische Operatoren	
`AND`	Logisches UND.
`OR`	Logisches ODER.
Weitere nützliche Operatoren	
`NOT`	Negation oder Verneinung
`IS NULL`	Prüft, ob die Spalte KEINEN Inhalt hat.
`NOT IS NULL`	Prüft, ob die Spalte irgendeinen Inhalt hat.
`LIKE`	Mustervergleich mit Texten im Zusammenhang mit JOKER-Zeichen `%` : beliebige Anzahl unbekannter Zeichen oder JOKER-Zeichen `_`: Ersatz für ein unbestimmtes Zeichen
`BETWEEN X AND Y`	Alle Werte im Intervall X bis Y inklusive X und Y.
`IN (Werte-Liste)`	Prüft, ob der Feldwert in der Werte-Liste ist.
`NOT IN (Werte-Liste)`	Prüft, ob der Feldwert NICHT in der Werte-Liste ist.

Funktionen verwenden

Jedes DBS bietet verschiedene Funktionen für numerische Berechnungen, Text-Operationen und Verarbeitung von Datum - und Zeit - Spalten. Häufig verwendete Funktionen werden auf dem Informationsblatt: *SQL, DQL, Ausdrücke und Funktionen* ausführlich erklärt und mit Beispielen veranschaulicht.

Kriterien richtig verwenden

Datentypen in Filterausdrücken berücksichtigen

Um eine gültige `WHERE`– oder `HAVING`–Klausel zu schreiben, müssen Sie die Datentypen der gefilterten Spalten genau kennen. Kriterien, die mit Text, Daten und Zeit arbeiten, werden in einfache Anführungszeichen gesetzt, alle anderen nicht.

Kriterien in einfache Anführungszeichen setzen bei	Keine Anführungszeichen bei
CHAR	DEC, ...
VARCHAR	INT, ...
DATE, TIME	
DATETIME, TIMESTAMP	
BLOB	

Wenn Kriterien einfache Anführungsstriche enthalten

Sollte in einem Suchbegriff ein einfaches Anführungszeichen vorkommen, muss es maskiert werden. Dabei gibt es zwei Möglichkeiten:

1. `SELECT * from bars WHERE name ='GROVER\'s Inn';`
2. `SELECT * from bars WHERE name ='GROVER''s Inn';`

Bei Möglichkeit 1 wird das Zeichen mit einem Backslash (\) maskiert. D.h. vor das Anführungszeichen wird ein Backslash eingefügt. Im zweiten Fall wird das Anführungszeichen gedoppelt.

Aliase - Spaltennamen und Tabellennamen temporär umbenennen

Mit Hilfe des Schlüsselwortes `AS` können Spalten und Tabellen in einem SELECT-Statement umbenannt werden. Mann nennt so einen geänderten Namen *Alias*. Die Aliase in einem SELECT-Statement gelten nur für das Statement in dem sie verwendet werden.

- Spalten werden umbenannt, damit der Benutzer das Ergebnis besser versteht.
- Tabellen werden in der Regel umbenannt, wenn ein SELECT-Statement mehrere Tabellen enthält. Ziel ist es, die Lesbarkeit für den Entwickler zu verbessern. Z.B. können lange Tabellennamen in kürzere Tabellennamen umbenannt werden. Das erhöht oft das Verständnis.

Hinweis: In einigen Fällen kann es vorkommen, dass Spalten in verschiedenen Tabellen den gleichen Namen haben - z.B. **`id`**. Die Spaltennamen sind also nicht eindeutig. In diesem Fall wird bei den identischen Spalten, der Tabellenname durch einen Punkt getrennt vorangestellt. Man nennt das einen *voll qualifizierten Spaltennamen*. Damit ist die Spalte genau festgelegt und es kann keine Verwechslung geben. Bei langen Tabellennamen kann das zu schwer lesbaren Statements führen. Hier schaffen Aliase Abhilfe.

```
SELECT m.m_id, vorname,nachname, link.m_id
       FROM mitglied AS m
                 INNER JOIN link_mitglied_sportart link
                 ON (m.m_id = link.m_id)
       WHERE m.m_id BETWEEN 100 AND 120
       ORDER BY m.m_id ASC;
```

Die SELECT-Anweisung im Einsatz

Für die folgenden SELECT-Beispiel werden folgende Tabellen aus der Datenbank *dbBundesliga2017* verwendet.

Tabelle: Spieler

Spieler_ID	Trikot_Nr	Spieler_Name	Land	Spiele	Tore	Vorlagen	Verein	Liga
1	1	Manuel Neuer	Deutschland	26	0	0	FC Bayern München	1
2	22	Tom Starke	Deutschland	3	0	0	FC Bayern München	1
3	26	Sven Ulreich	Deutschland	5	0	0	FC Bayern München	1
4	27	David Alaba	Österreich	32	4	0	FC Bayern München	1
5	18	Juan Bernat	Spanien	18	2	0	FC Bayern München	1
6	17	Jerome Boateng	Deutschland	13	0	0	FC Bayern München	1
7	39	Nicolas Feldhahn	Deutschland	0	0	0	FC Bayern München	1
8	13	Rafinha	Brasilien	20	1	0	FC Bayern München	1
9	20	Felix Gotze	Deutschland	0	0	0	FC Bayern München	1

Untersuchen Sie die folgenden SELECT – Statements und erklären Sie kurz, welche Daten nach der Ausführung angezeigt werden.

Aufgabe 1:

```
SELECT * from Spieler;
```

Aufgabe 2:

```
SELECT * from Spieler WHERE Spieler_Name = 'Manuel Neuer' ;
```

Aufgabe 3:

```
SELECT * from Spieler WHERE Liga = 1 ;
```

Aufgabe 4:

```
SELECT * from Spieler WHERE Spiele >= 18 ;
```

Aufgabe 5:

```
SELECT Spieler_Name,Spiele, Tore FROM Spieler
          WHERE Verein <> 'FC Bayern München';
```

Aufgabe 6:

```
SELECT Spieler_Name,Spiele, Tore FROM Spieler
          WHERE Verein = 'FC Bayern München' AND  Spiele <=10 ;
```

Aufgabe 7:

```
SELECT Spieler_Name,Spiele, Tore FROM Spieler
          WHERE (Tore < 3 AND Vorlagen > 0) OR (Tore > 5) ;
```

Aufgabe 8:

```
SELECT Trikot_Nr,Spieler_Name,Spiele, Verein FROM Spieler
          WHERE Tore BETWEEN 5 AND 10
          ORDER BY VEREIN;
```

Aufgabe 9

```
SELECT Trikot_Nr,Spieler_Name,Spiele FROM Spieler
          WHERE NOT Tore BETWEEN 5 AND 10 AND
                NOT Verein = 'FC Bayern München';
```

Wir wechseln jetzt die Tabelle...

Tabelle: Spiele

Spiel_id	Spieltag	Datum	Uhrzeit	Verein_Heim	Verein_Gast	Tore_Heim	Tore_Gast
497	1	2016-08-07	15:30:00	1. FC Heidenheim 1846	FC Erzgebirge Aue	1	0
509	3	2016-08-26	18:30:00	1. FC Heidenheim 1846	Kickers Würzburg	1	2
530	5	2016-09-17	13:00:00	1. FC Heidenheim 1846	Fortuna Düsseldorf	2	0
547	7	2016-09-24	13:00:00	1. FC Heidenheim 1846	1. FC Kaiserslautern	3	0
558	8	2016-10-02	13:30:00	1. FC Heidenheim 1846	Eintracht Braunschweig	1	1
575	10	2016-10-22	13:00:00	1. FC Heidenheim 1846	Dynamo Dresden	0	0
589	12	2016-11-04	18:30:00	1. FC Heidenheim 1846	Karlsruher SC	2	1
610	14	2016-11-26	13:00:00	1. FC Heidenheim 1846	FC St. Pauli	2	0
627	16	2016-12-09	18:30:00	1. FC Heidenheim 1846	1. FC Union Berlin	3	0

Aufgabe 10:

```
SELECT * FROM Spiele
          WHERE Datum = '2016-08-07'
          ORDER BY Verein_Heim ASC;
```

Aufgabe 11:

```
SELECT Spieltag, Datum, Verein_Heim, Verein_Gast, Tore_Heim, Tore_Gast,
       ABS(Tore_Heim -Tore_Gast) AS Tordifferenz FROM Spiele
       WHERE YEAR(Datum) = 2016
       ORDER BY Spieltag ASC;
```

Aufgabe 12:

```
SELECT * FROM Spiele
       WHERE Verein_Gast LIKE 'K%'
       ORDER BY Datum DESC, Uhrzeit ASC;
```

Aufgabe 13:

```
SELECT * FROM Spiele
       WHERE Verein_Heim LIKE '%FC%' OR Verein_Gast LIKE '%FC%'
       ORDER BY Spieltag, Uhrzeit;
```