SQL-Sprachvielfalt

DML Data Manipulation Language: Daten einfügen, ändern und löschen

DQL Data Query Language: Daten abfragen

DDL Data Definition Language: Tabellen und andere Strukturelemente erzeugen, ändern, löschen

Grundsätze

- Die Reihenfolge der Schlüsselwörter ist immer gleich.
- Groß- und Kleinschreibung wird nur innerhalb von Anführungszeichen beachtet.
- Außerhalb von Anführungszeichen können beliebig viele Leerzeichen und Zeilenumbrüche verwendet werden, wenn ein Leerzeichen zulässig ist.
- INSERT, UPDATE und DELETE wirken immer nur auf *eine* Tabelle.

Beispieltabelle

Kunde		
Kundennummer	*Kundenname*	*Ort*
INTEGER Primärschlüssel	VARCHAR(40)	VARCHAR(30) Standard: Pforzheim

Dazu kommt noch die gleich aufgebaute Tabelle Kunde2

SQL-Befehle zum Bearbeiten von Daten

INSERT Daten einfügen

UPDATE Daten ändern

DELETE Daten löschen

INSERT: Standardfall

Alle Felder belegen, Einfügen in Spaltenreihenfolge

```
INSERT INTO Kunde VALUES
(10, 'Fred', 'Remchingen')
```

Kunde		
Kundennummer	*Kundenname*	*Ort*
10	Fred	Remchingen

- Felder mit Komma trennen
- Text in einfache Anführungszeichen einschließen

INSERT: DEFAULT

Standardwerte für einzelne Felder:

```
INSERT INTO Kunde VALUES (11, 'Pforzheimerin', DEFAULT)
```

Kunde		
Kundennummer	*Kundenname*	*Ort*
10	Fred	Remchingen
11	Pforzheimerin	Pforzheim

Alternativformulierung:

```
INSERT INTO Kunde (Kundennummer, Kundenname)
VALUES (11, 'Pforzheimerin')
```


INSERT: DEFAULT

Standardwerte für einzelne Felder:

```
INSERT INTO Kunde VALUES (11, 'Pforzheimerin', DEFAULT)
```

Kunde		
Kundennummer	*Kundenname*	*Ort*
10	Fred	Remchingen
11	Pforzheimerin	Pforzheim

Alternativformulierung:

```
INSERT INTO Kunde (Kundennummer, Kundenname)
VALUES (11, 'Pforzheimerin')
```

- DEFAULT ohne Anführungszeichen weist die Datenbank an, den Standardwert zu verwenden (in diesem Fall 'Pforzheim')
- alternativ dazu können die zu füllenden Felder aufgezählt werden, andere Felder werden dann mit Standardwert belegt (falls angelegt, sonst mit NULL).

INSERT: NULL

Undefinierte Eingabe für einzelne Felder:

```
INSERT INTO Kunde VALUES
(12, 'Mister X', NULL)
```

Kunde		
Kundennummer	*Kundenname*	*Ort*
10	Fred	Remchingen
11	Pforzheimerin	Pforzheim
12	Mister X	?

INSERT: NULL

Undefinierte Eingabe für einzelne Felder:

```
INSERT INTO Kunde VALUES
(12, 'Mister X', NULL)
```

Kunde		
Kundennummer	*Kundenname*	*Ort*
10	Fred	Remchingen
11	Pforzheimerin	Pforzheim
12	Mister X	?

- `NULL` ohne Anführungszeichen weist die Datenbank an, den Eintrag unbestimmt zu lassen
- alternativ dazu können die zu füllenden Felder aufgezählt werden, andere Felder werden dann mit NULL belegt (falls es keinen Standardwert gibt).

INSERT: Leereingabe

Leereingabe für einzelne Felder:

```
INSERT INTO Kunde VALUES
(15, 'Vagabund', '')
```

Kunde		
Kundennummer	*Kundenname*	*Ort*
...	...	
12	Mister X	?
15	Vagabund	

INSERT: Leereingabe

Leereingabe für einzelne Felder:

```
INSERT INTO Kunde VALUES
(15, 'Vagabund', '')
```

Kunde		
Kundennummer	*Kundenname*	*Ort*
...	...	
12	Mister X	?
15	Vagabund	

- Die Eingabe eines leeren Textes (zwei Apostrophen nacheinander) ist nicht das gleiche wie die Eingabe von NULL: Der Vagabund hat keinen Wohnort, bei Mister X wissen wir den Wohnort nicht.

INSERT: Reihenfolge ändern

Felder in der Reihenfolge aufzählen, in der sie angegeben werden.
Weglassen einzelner Felder führt zum Auffüllen mit Standardwerten

```
INSERT INTO Kunde (Ort, Kundenname, Kundennummer)
VALUES ('Ispringen', 'Heinrich', 13)
```

Kunde		
Kundennummer	*Kundenname*	*Ort*
...	...	...
13	Heinrich	Ispringen
14	Pforzheimerin	Pforzheim

```
INSERT INTO Kunde (Kundenname, Kundennummer) VALUES
('Pforzheimerin', 14)
```


INSERT: Einfügen aus Abfrage

```
INSERT INTO Kunde2
SELECT * FROM Kunde
```

Kunde2		
Kundennummer	*Kundenname*	*Ort*
10	Fred	Remchingen
...	...	...

So können Daten aus anderen Tabellen kopiert werden.

INSERT: Einfügen aus Abfrage

```
INSERT INTO Kunde2
SELECT Kundennummer+100, Kundenname, Ort FROM Kunde
WHERE Ort <> 'Pforzheim'
```

Kunde2		
Kundennummer	*Kundenname*	*Ort*
...	...	...
110	Fred	Remchingen
...	...	...

DELETE: Tabelle leeren

```
DELETE FROM Kunde2
```

Kunde2		
Kundennummer	*Kundenname*	*Ort*

- DELETE ohne Einschränkungen leert eine Tabelle vollständig
- Die Tabellenstruktur bleibt erhalten

Füllen Sie die Tabelle anschließend mit
`INSERT INTO Kunde2 SELECT * FROM Kunde`
wieder auf.

DELETE: Löschen mit Bedingungen

```
DELETE FROM Kunde2
WHERE Ort <> 'Pforzheim'
```

Kunde2		
Kundennummer	*Kundenname*	*Ort*
11	Pforzheimerin	Pforzheim
12	Mister X	?

DML: Daten einfügen, ändern, löschen Delete

DELETE: Löschen mit Bedingungen

DELETE FROM Kunde2
WHERE Ort <> 'Pforzheim'

Kunde2		
Kundennummer	*Kundenname*	*Ort*
11	Pforzheimerin	Pforzheim
12	Mister X	?

Alle Kunden, bei denen der Ort bekannt und nicht 'Pforzheim' ist, werden gelöscht.

Peter Willadt (Ludwig-Erhard-Schule) SQL-Befehle 2019-11 13 / 49

UPDATE: Fehlanwendung

```
UPDATE Kunde2 SET Kundenname='Fred'
```

- UPDATE ohne einschränkende Bedingungen ändert *jede* Zeile einer Tabelle
- Nach diesem Befehl heißen alle Kunden »Fred«

DML: Daten einfügen, ändern, löschen Update

UPDATE: Standardbeispiel

```
UPDATE Kunde2 SET Ort='Wilferdingen'
WHERE Ort='Remchingen'
```

- Alle Kunden, die »Remchingen« als Ort eingetragen hatten, wohnen jetzt in »Wilferdingen«
- NULL verlangt eine spezielle Formulierung

```
UPDATE Kunde2 SET Ort='unbekannt'
WHERE Ort IS NULL
```

Peter Willadt (Ludwig-Erhard-Schule) SQL-Befehle 2019-11 15 / 49

UPDATE: Mehrere Felder ändern

```
UPDATE Kunde2 SET Ort='Gustav', Kundenname='Ispringen'
WHERE Kundennummer=11
```

Kunde Nummer 11 heißt jetzt »Ispringen« und wohnt in »Gustav«.
Das lässt sich korrigieren:

```
UPDATE Kunde2 SET Ort=Kundenname, Kundenname=Ort
WHERE Kundennummer=11
```


DML: Daten einfügen, ändern, löschen Update

UPDATE: Berechnungen

```
UPDATE Kunde2 SET Kundennummer=2*Kundennummer
```

So werden die Kundennummern verdoppelt.
Außer den Grundrechenarten stehen auch Funktionen zur Verfügung.

```
UPDATE Kunde2 SET Ort=UPPER(Ort)
```

Jetzt sind die Wohnorte komplett in Großbuchstaben umgewandelt.

Peter Willadt (Ludwig-Erhard-Schule) SQL-Befehle 2019-11 17 / 49

DML und Views

DML-Befehle lassen sich auch auf *Views* statt Tabellen anwenden, falls die Views bestimmte Bedingungen erfüllen. Die Bedingungen für DELETE sind am einfachsten zu erfüllen, die für INSERT am schwierigsten.
Beispiel: Wenn im View kein Primärschlüssel enthalten ist, können Sie nichts einfügen.
Einige Datenbanksysteme setzen DML auf Views nicht korrekt um. Es ist daher sicherer, DML nur auf Tabellen anzusetzen.

DQL

- Die Data Query Language (DQL) besteht nur aus dem Befehl SELECT.
- Er ist der komplexeste und am häufigsten eingesetzte SQL-Befehl.

Beispieltabelle: Tabellenaufbau

Kunde		
Kundennummer	*Kundenname*	*Ort*
INTEGER Primärschlüssel	VARCHAR(40)	VARCHAR(30) Standard: Pforzheim

Beispieltabelle: Tabelleninhalt

Kunde		
Kundennummer	*Kundenname*	*Ort*
10	Fred	Remchingen
11	Pforzheimerin	Pforzheim
12	Mister X	?
13	Heinrich	Ispringen
14	Pforzheimerin	Pforzheim
15	Vagabund	

Spalten auswählen

`SELECT * FROM Kunde`
Alle Spalten werden in der Reihenfolge, in der sie angelegt sind, angezeigt.
`SELECT Kundenname, Ort FROM Kunde`
Kundenname und Ort werden angezeigt. Die Zeilenreihenfolge ist unbestimmt.

Kundenname	*Ort*
Fred	Remchingen
Pforzheimerin	Pforzheim
Mister X	?
Heinrich	Ispringen
Pforzheimerin	Pforzheim
Vagabund	

Rechnen

`SELECT Kundennummer*5 FROM Kunde`
Die mit 5 multiplizierte Kundennummer wird angezeigt. Zeilenreihenfolge ist unbestimmt.

Expression1
50
55
60
65
70
75

Zusätzlich zu den Grundrechenarten gibt es noch Zeilenfunktionen, zum Beispiel `ROUND()` oder `UPPER()`

Spalten umbenennen

`SELECT Ort AS Wohnort FROM Kunde`

Wohnort
Remchingen
Pforzheim
?
Ispringen
Pforzheim

Anstatt »Ort« steht jetzt »Wohnort« als Überschrift über der Ergebnistabelle.

Konstante Texte einfügen

`SELECT 'Herrn', Kundenname FROM Kunden`

Expression1	*Kundenname*
Herrn	Fred
Herrn	Pforzheimerin
Herrn	Mister X
Herrn	Heinrich
Herrn	Pforzheimerin

Texte zusammenfügen

`SELECT 'Herrn ' || Kundenname FROM Kunden`

Expression1
Herrn Fred
Herrn Pforzheimerin
Herrn Mister X
Herrn Heinrich
Herrn Pforzheimerin

- Durch die Verkettung stehen die beiden Texte in *einer* Spalte
- Dieses Beispiel funktioniert nicht mit MySQL

Aggregatfunktionen

Zeige Mittelwert und Summe der Kundennummern und die Anzahl der Zeilen in der Kundentabelle:

```
SELECT AVG(Kundennummer), SUM(Kundennummer),
MAX(Kundennummer),
COUNT(*) FROM Kunde
```

Expression1	*Expression2*	*Expression3*	*Expression4*
12	60	14	5

Aggregatfunktionen

Zeige Mittelwert und Summe der Kundennummern und die Anzahl der Zeilen in der Kundentabelle, mit sinnvollen Überschriften:

```
SELECT AVG(Kundennummer) AS Schnitt, SUM(Kundennummer)
as Summe
MAX(Kundennummer) as Maximum, COUNT(*) as Anzahl
FROM Kunde
```

SCHNITT	*SUMME*	*MAXIMUM*	ANZAHL
12	60	14	5

Zusammen mit `GROUP BY` lassen sich die Daten gegliedert auswerten.

Gruppierung

Zeige Anzahl Kunden je Ort:

```
SELECT Ort, COUNT(*) as Anzahl
FROM Kunde GROUP BY Ort ORDER BY Ort
```

ORT	*ANZAHL*
	1
Ispringen	1
Pforzheim	2
Remchingen	1

Tabellen auswählen

- Tabellenauswahl findet im `FROM`-Teil des `SELECT`-Befehles statt.
- Wenn mehrere Tabellen angegeben werden, müssen sie mit Komma getrennt aufgezählt werden.
- Wenn die Tabellen nicht über `JOIN` oder `WHERE` verknüpft sind, erzeugt die Datenbank das *kartesische Produkt*: Jede Zeile jeder Tabelle wird mit jeder Zeile jeder anderen Tabelle kombiniert.
- Gleichnamige Spalten in unterschiedlichen Tabellen müssen durch Voranstellen des Tabellennamens und eines Punktes eindeutig gemacht werden.

Beispieltabellen

Mann

MannID	*Vorname*	*heiratet*
1	Bernd	
2	Oskar	3
3	Fred	1

Frau

FrauID	*Vorname*
1	Sabine
3	Susanne
5	Erika

Kartesisches Podukt

```
SELECT Mann.Vorname, Frau.Vorname FROM Mann, Frau
```

Vorname	*Vorname*
Bernd	Sabine
Oskar	Sabine
Fred	Sabine
Bernd	Susanne
Oskar	Susanne
Fred	Susanne
Bernd	Erika
Oskar	Erika
Fred	Erika

Inner Join

Ein `INNER JOIN` verknüpft *die* Zeilen der beiden Tabellen miteinander, bei denen der Inhalt der angegebenen Felder gleich ist.

```
SELECT Mann.Vorname, Frau.Vorname
FROM Mann INNER JOIN Frau ON Mann.heiratet=Frau.FrauID
```

Vorname	*Vorname*
Oskar	Susanne
Fred	Sabine

Umbenennen der Tabellen macht den Ausdruck kürzer:

```
SELECT M.Vorname, F.Vorname
FROM Mann M INNER JOIN Frau F ON M.heiratet=F.FrauID
```

Das Ergebnis bleibt gleich.

WHERE statt JOIN

Anstelle einer `Join`-Bedingung im `FROM`-Teil der Abfrage kann die Verknüpfung auch im `WHERE`-Teil erfolgen.

```
SELECT M.Vorname, F.Vorname
FROM Mann M, Frau F
WHERE M.heiratet=F.FrauID
```

Vorname	*Vorname*
Oskar	Susanne
Fred	Sabine

Das Ergebnis ist identisch mit dem, das mit `INNER JOIN` erhalten wurde, allerdings bietet `WHERE` noch weitere Möglichkeiten.

Left Join

```
SELECT M.Vorname, F.Vorname
FROM Mann M LEFT JOIN Frau F ON M.heiratet=F.FrauID
```

Vorname	*Vorname*
Bernd	?
Oskar	Susanne
Fred	Sabine

Ein `LEFT JOIN` enthält alle Zeilen der ersten Tabelle und nur die Zeilen der zweiten Tabelle, die zu den Bedingungen passen. In den übrigen Zeilen steht an deren Stelle `NULL`.

Right Join

```
SELECT M.Vorname, F.Vorname
FROM Mann M RIGHT JOIN Frau F ON M.heiratet=F.FrauID
```

Vorname	*Vorname*
?	Erika
Oskar	Susanne
Fred	Sabine

Ein `RIGHT JOIN` enthält alle Zeilen der zweiten Tabelle und nur die Zeilen der ersten Tabelle, die zu den Bedingungen passen. In den übrigen Zeilen steht an deren Stelle `NULL`.

Full outer Join

```
SELECT M.Vorname, F.Vorname
FROM Mann M FULL OUTER JOIN Frau F ON
M.heiratet=F.FrauID
```

Vorname	*Vorname*
Bernd	?
?	Erika
Oskar	Susanne
Fred	Sabine

Ein `FULL OUTER JOIN` enthält alle Zeilen beider Tabellen und kombiniert sie wo möglich. Übrige Felder werden mit `NULL` aufgefüllt.

Zeilen auswählen

- Die Auswahl der Zeilen, die für die Erstellung des Ergebnisses benötigt werden, erfolgt mit dem Schlüsselwort `WHERE`.
- Eine mit `WHERE` für die Auswahl benötigte Spalte *muss nicht* als Ergebnisspalte verwendet werden.

Einfacher Vergleich

```
SELECT * FROM Mann
WHERE heiratet = 1
```

MannID	*Vorname*	*heiratet*
3	Fred	1

```
SELECT * FROM Mann
WHERE heiratet < 1
```

MannID	*Vorname*	*heiratet*

```
SELECT * FROM Mann
WHERE heiratet > 1
```

MannID	*Vorname*	*heiratet*
2	Oskar	3

```
SELECT * FROM Mann
WHERE heiratet <> 1
```

MannID	*Vorname*	*heiratet*
2	Oskar	3

```
SELECT * FROM Mann
WHERE heiratet >= 1
```

MannID	*Vorname*	*heiratet*
2	Oskar	3
3	Fred	1

```
SELECT * FROM Mann
WHERE heiratet IS NULL
```

MannID	*Vorname*	*heiratet*
1	Bernd	?

```
SELECT * FROM Mann
WHERE heiratet IS NOT NULL
```

MannID	*Vorname*	*heiratet*
2	Oskar	3
3	Fred	1

Text-Vergleich

- Text muss in einfache Anführungszeichen
- Groß- und Kleinschreibung wird unterschieden
- Mit `UPPER()` vergleichen Sie Großschreibungs-unabhängig

```
SELECT * FROM Mann
WHERE Vorname = 'Fred'
```

MannID	*Vorname*	*heiratet*
3	Fred	1

```
SELECT * FROM Mann
WHERE Vorname = 'fred'
```

MannID	*Vorname*	*heiratet*

```
SELECT * FROM Mann
WHERE Vorname = 'FRED'
```

MannID	*Vorname*	*heiratet*

```
SELECT * FROM Mann
WHERE UPPER(Vorname) =
'FRED'
```

MannID	*Vorname*	*heiratet*
3	Fred	1

```
SELECT * FROM Mann
WHERE UPPER(Vorname) =
UPPER('fred')
```

MannID	*Vorname*	*heiratet*
3	Fred	1

Text-Mustervergleich

- Mustervergleich wird mit dem Schlüsselwort `LIKE` aktiviert
- Das Prozentzeichen steht als Platzhalter (Joker) für beliebig viele Zeichen
- Der Unterstrich steht als Platzhalter (Joker) für genau ein Zeichen

```
SELECT * FROM Mann
WHERE Vorname LIKE 'F%'
```

MannID	*Vorname*	*heiratet*
3	Fred	1

```
SELECT * FROM Mann
WHERE Vorname LIKE 'F_ed'
```

MannID	*Vorname*	*heiratet*
3	Fred	1

```
SELECT * FROM Frau
WHERE Vorname LIKE 'S%e'
```

FrauID	*Vorname*
1	Sabine
3	Susanne

```
SELECT * FROM Mann
WHERE Vorname LIKE 'f%'
```

MannID	*Vorname*	*heiratet*

Sortieren

- Wenn keine Sortierung angegeben wird, werden die Daten in unbestimmter Reihenfolge geliefert.
- Sortiert wird mit `ORDER BY`.
- Die Spalte, nach der sortiert wird, muss nicht angezeigt werden.
- Falls nach mehreren Spalten sortiert werden muss, die Spaltennamen mit Komma trennen.
- Absteigend sortiert wird mit dem Schlüsselwort `DESC`.
- Funktionen und Berechnungen können verwendet werden.
- Spalten können auch mit ihrer Nummer statt ihrem Namen angesprochen werden.

Sortieren

```
SELECT * FROM Mann
ORDER BY Vorname
```

MannID	*Vorname*	*heiratet*
1	Bernd	
3	Fred	1
2	Oskar	3

```
SELECT Vorname FROM Mann
ORDER BY MannID
```

Vorname
Bernd
Oskar
Fred

```
SELECT * FROM Mann
ORDER BY 2 DESC
```

MannID	*Vorname*	*heiratet*
2	Oskar	3
3	Fred	1
1	Bernd	

```
SELECT * FROM Mann
ORDER BY UPPER(Vorname)
```

MannID	*Vorname*	*heiratet*
1	Bernd	
3	Fred	1
2	Oskar	1

Aggregatfunktionen

- Aggregatfunktionen oder *Spaltenfunktionen* fassen mehrere Werte einer Spalte zusammen. Typische Beispiele: `AVG()` (Mittelwert), `SUM()` (Summe), `COUNT()` (Anzahl)
- Wird `GROUP BY` mit angegeben, wird die Zusammenfassung gegliedert.
- Ohne `GROUP BY` dürfen im `SELECT`-Teil der Abfrage nur Spaltenfunktionen verwendet werden.

Beispieltabelle

Lieferung

Kundennummer	*Ort*	*Umsatz*
100	Karlsruhe	200
150	Karlsruhe	50
250	Pforzheim	50
300	Pforzheim	100
350	Pforzheim	200

Summen bilden

Zeige den Gesamt-Umsatz:

```
SELECT SUM(Umsatz)
FROM Lieferung
```

Expression1
600

Zeige den Umsatz in Karlsruhe:

```
SELECT SUM(Umsatz)
FROM Lieferung
WHERE Ort = 'Karlsruhe'
```

Expression1
250

Zeige den Umsatz je Ort:

```
SELECT Ort, SUM(Umsatz)
FROM Lieferung
GROUP BY Ort
```

Ort	*Expression1*
Karlsruhe	250
Pforzheim	350

HAVING

Mit `HAVING` können Bedingungen angegeben werden, die *nach* der Gruppenbildung abgearbeitet werden.

```
SELECT Ort, SUM(Umsatz)
FROM Lieferung
GROUP BY Ort
```

Ort	*Expression1*
Karlsruhe	250
Pforzheim	350

Zeige den Umsatz je Ort. Dieses Beispiel war schon auf der vorigen Folie.

```
SELECT Ort, SUM(Umsatz)
FROM Lieferung
GROUP BY Ort
HAVING SUM(Umsatz) > 300
```

Ort	*Expression1*
Pforzheim	350

Zeige den Umsatz je Ort, aber nur für alle Orte mit mehr als 300 Umsatz

Unterabfragen

- Im `WHERE`-Teil einer Abfrage kann eine andere Abfrage ausgeführt werden.
- Diese Abfrage kann auch auf die Hauptabfrage Bezug nehmen (korrelierte Unterabfrage)

Zeige alle Kunden mit überdurchschnittlichem Umsatz

```
SELECT Kundennummer,Umsatz
FROM Lieferung
WHERE Umsatz >
SELECT AVG(Umsatz) FROM Lieferung
```

Kundennummer	*Umsatz*
100	200
350	200

Unterabfragen

Zeige alle Frauen, die nicht verheiratet sind:

```
SELECT * FROM Frau WHERE FrauID NOT IN
(SELECT heiratet FROM Mann)
```

Frau

FrauID	*Vorname*
5	Erika