# Datenbanken

*Marvin Baeumer* **2023-12-06 10:20**

---

## SQL

1. **SELECT:**

```
SELECT [Spalte1], [Spalte2] FROM [Tabelle]
```

> `SELECT` ist der grundlegende Befehl für Datenabfragen. Man wählt Spalten aus einer oder mehreren Tabellen aus, um Daten abzurufen.

2. **DISTINCT:**

```
SELECT DISTINCT [Spalte] FROM [Tabelle]
WHERE [Bedingung]
```

> `DISTINCT` wird zusammen mit `SELECT` verwendet, um eindeutige Werte in einer bestimmten Spalte zurückzugeben. Dadurch werden doppelte Werte eliminiert, und nur verschiedene Werte werden angezeigt.

1. **ORDER BY:**

```
SELECT [Spalte1], [Spalte2] FROM [Tabelle]
ORDER BY [Spalte] ASC|DESC
```

> `ORDER BY` wird verwendet, um die Ergebnisse der Abfrage zu sortieren. Man gibt die Spalte an, nach der die Sortierung erfolgen soll, und optional die Sortierreihenfolge (`ASC` für aufsteigend, `DESC` für absteigend).

4. **WHERE:**

```
SELECT [Spalte1], [Spalte2] FROM [Tabelle]
WHERE [Bedingung]
```

> `WHERE` wird verwendet, um Bedingungen für die Datenauswahl festzulegen. Nur die Datensätze, die die angegebenen Bedingungen erfüllen, werden in den Ergebnissen angezeigt.

5. **GROUP BY:**

```
SELECT [Spalte1], [Spalte2] FROM [Tabelle]
GROUP BY [Spalte1]
```

> `GROUP BY` wird verwendet, um Datensätze basierend auf einer oder mehreren Spalten zu gruppieren und Duplikate zu eliminieren.

6. **HAVING:**

```
SELECT [Spalte], COUNT(*) AS Anzahl FROM [Tabelle]
GROUP BY [Spalte]
HAVING Anzahl > Wert
```

> `HAVING` wird nach `GROUP BY` verwendet, um Bedingungen für gruppierte Daten festzulegen. Es filtert die Ergebnisse basierend auf aggregierten Funktionen.

7. **CREATE TABLE**

```
CREATE TABLE Mitarbeiter (
[Spalte1] INTEGER NOT NULL,
[Spalte2] TEXT NOT NULL,
[Spalte3] TEXT NOT NULL,
PRIMARY KEY ([Spalte1]),
FOREIGN KEY (Spalte2) REFRENCES [Tabelle]([Spalte]),
);
```

8. **INSERT:**

```
INSERT INTO [Tabelle] ([Spalte1], [Spalte2])
VALUES (Wert1, Wert2);
```

> Der `INSERT`-Befehl wird verwendet, um neue Datensätze in eine Tabelle einzufügen. Man gibt dabei die Werte für jede Spalte an, die mit den definierten Spalten der Tabelle übereinstimmen. Diese Operation ermöglicht das Hinzufügen von Daten in die Datenbank.

9. **UPDATE:**

```
UPDATE [Tabelle] SET
[Spalte1] = Wert1,
[Spalte2] = Wert2 WHERE [Bedingung];
```

> Mit dem `UPDATE`-Befehl werden bestehende Datensätze in einer Tabelle aktualisiert. Man gibt die zu aktualisierenden Spalten und die entsprechenden Werte an. Die `WHERE`-Klausel dient dazu, den Bereich der zu aktualisierenden Datensätze zu beschränken, um sicherzustellen, dass nur bestimmte Datensätze modifiziert werden.

10. **DELETE:**

```
DELETE FROM [Tabelle] WHERE [Bedingung];
```

> Der `DELETE`-Befehl entfernt Datensätze aus einer Tabelle, die eine bestimmte Bedingung erfüllen. Wie beim `UPDATE` ermöglicht die `WHERE`-Klausel das Festlegen von Bedingungen, um den Bereich der zu löschenden Datensätze zu steuern. Vorsicht ist geboten, da dieser Befehl dauerhafte Datenlöschungen verursacht.

11. **BETWEEN:**

```
SELECT [Spalte] FROM [Tabelle]
WHERE [Spalte] BETWEEN Wert1 AND Wert2;
```

> `BETWEEN` wird verwendet, um Datensätze basierend auf einem bestimmten Wertebereich in einer Spalte auszuwählen. In der Abfrage werden nur die Datensätze zurückgegeben, deren Werte in der angegebenen Spalte zwischen Wert1 und Wert2 liegen.

# ER-Modell

> Ein ER-Modell besteht meistens aus einem oder mehreren Entitätstypen. Diese Entitätstypen können Attribute besitzen. Attribute sind Eigenschaften eines Entitätstyps, das heißt, ein Entitätstyp "Kunde" könnte die Attribute KundenNr, Vorname und Nachname enthalten. Entitätstypen können in Beziehung zu anderen Entitätstypen stehen. Dabei wird das Verhältnis zwischen den beiden Entitätstypen mit Kardinalitäten dargestellt. Zum Beispiel könnte die Tabelle "Kunde" in einer 1:n-Beziehung mit dem Entitätstyp "Produkt" stehen. *Ein Kunde kauft ein oder mehrere Produkte. Ein Produkt wird genau von einem Kunden gekauft.* Bei einem ER-Modell werden Primärschlüssel $\underline{unterstrichen}$.

# Kardinalitäten

| NC-Notation | Verhältnis |
|---|---|
| 1 | genau einer |
| c | keiner oder einer |
| n | einer oder mehrere |
| nc | keine, einer oder mehrere |

> Wichtig bei einer $n:m$ oder $nc:mc$ oder $n:mc$ Beziehung ersetzt man das eine 'n' durch ein 'm'.

# Relationsschema

> Ein Relationschema ist eine formale Darstellung einer Datenbanktabelle. Es definiert die Struktur der Tabelle, indem es die Namen der Spalten (Attribute) angibt und Fremd und Primaerschluessel kennzeichnet.

Kunde($\underline{KundenNr}$, Name, Vorname)

produkt($\underline{ProduktNr}$, Bezeichnung)

kauft($\underline{Datum}, \uparrow \underline{KundenNr}, \uparrow \underline{ProduktNr}$)

# Normalform

1. Normalform:

> Eine Relation befindet sich in der ersten Normalform, wenn alle Attribute nur einfache Attributswerte aufweisen (Bezeichnung: atomar $\rightarrow$ nicht teilbar). Alles muss einzeln stehen. Der Primärschlüssel wird nicht mehr eindeutig, also muss er zusammengesetzt werden und aus zwei oder mehr Teilen bestehen.

---

2. Normalform: Eine Relation befindet sich in der zweiten Normalform, wenn:

> Sie in der ersten Normalform ist. Jedes Nicht-Schlüssel-Attribut vom Primärschlüssel voll funktional abhängig ist.

---

3. Normalform: Eine Relation befindet sich in der dritten Normalform, wenn:

> Sie in der zweiten Normalform ist. Jedes Nichtschlüsselattribut nicht transitiv vom Primärschlüssel abhängig ist.

# Begriffe

## Redundanzen

> Redundanzen bezeichnen die unnötige Wiederholung von Daten in einer Datenbank, was zu ineffizientem Speicherplatzverbrauch führen kann. Sie sollten durch effektives Datenbankdesign und Normalisierung minimiert werden.

## Dateninkonsistenz

> Dateninkonsistenz tritt auf, wenn Daten in einer Datenbank widersprüchlich oder nicht übereinstimmend sind. Dies kann durch fehlende Referenzielle Integrität, Redundanzen oder unsaubere Aktualisierungsprozesse entstehen. Dateninkonsistenz sollte vermieden werden, um die Genauigkeit und Zuverlässigkeit der Daten sicherzustellen.

## Referenzielle Integrität

> Referenzielle Integrität gewährleistet konsistente und gültige Beziehungen zwischen Tabellen in einer Datenbank. Durch die korrekte Verwendung von Fremdschlüsseln wird sichergestellt, dass jeder Fremdschlüssel auf einen existierenden Primärschlüssel in einer anderen Tabelle verweist. Dies schützt vor inkonsistenten oder ungültigen Beziehungen und trägt zur Datenkonsistenz und -zuverlässigkeit bei.

## Transitiv

> In Bezug auf Datenbankdesign bezieht sich "transitiv" auf eine spezifische Form der funktionalen Abhängigkeit zwischen Attributen. Eine transitive Abhängigkeit tritt auf, wenn ein Nicht-Schlüsselattribut von einem anderen Nicht-Schlüsselattribut abhängt, das selbst von einem Primärschlüsselattribut abhängt.