Die Syntax des SELECT-Befehls:

```
SELECT
   [ DISTINCT | ALL ]
     <Spaltenliste>
     FROM <Tabelle>
   [ WHERE-Klausel ]
   [ GROUP BY-Klausel ]
   [ HAVING-Klausel ]
   [ UNION [ALL] SELECT-Befehl ]
   [ ORDER BY-Klausel ];
```

Aggregatsfunktionen:

- **SUM:** SUM = Summe. Summiert die Werte in einer Spalte (nur für numerische Datentypen).
- **AVG:** AVG = Average = Durchschnitt. Gibt den Mittelwert der Werte in einer Spalte zurück (nur für numerische Datentypen).
- **COUNT:** COUNT = zählen. Gibt die Anzahl von Datensätzen zurück.
- **COUNT DISTINCT:** Gibt die Anzahl von eindeutigen / unterschiedlichen Werten zurück.
- **MIN:** MIN = Minimum. Gibt den kleinsten Wert in einer Gruppe zurück.
- **MAX:** MAX = Maximum. Gibt den grössten Wert in einer Gruppe zurück.

Befehle:

- **SELECT** = Daten auswählen
- **INSERT** = Daten einfügen
- **UPDATE** = Daten aktualisieren / ändern
- **DELETE** = Daten löschen

```
SELECT Vorname, Nachname, Geburtsdatum FROM Lernende LIMIT 10;
```

```
SELECT Vorname || ' ' || Nachname AS 'Name', Strasse, Plz, Ort
  FROM Lernende;
```

```
SELECT * FROM Lernende ORDER BY Ort, Nachname;
```

```
SELECT * FROM Lernende ORDER BY Ort ASC, Nachname ASC;
```

➔ Identische Abfrage wir oben, nur dass die aufsteigende Sortierreihenfolge explizit angegeben wird.

```
SELECT * FROM Lernende ORDER BY Ort DESC;
```

➔ Es werden alle Lernenden ausgegeben, und zwar nach Ort absteigend sortiert.

```
SELECT * FROM Lernende WHERE Ort = "Berlin";
```

```
SELECT MIN(Jahrgang), MAX(Jahrgang) FROM Lernende;
```

Runden Sie das Durchschnittsalter auf 2 Stellen nach dem Komma.

```
SELECT ROUND (AVG (2021 - jahrgang), 2)
  AS "Durchschnittsalter" FROM lernende;
```

oder

```
SELECT ROUND (AVG (strftime('%Y','now') - jahrgang), 2
  AS "Durchschnittsalter" FROM lernende;
```

Aufgabe 6.7

Geben Sie die totale Anzahl Lernenden mit einem entsprechenden Titel aus.

```
SELECT COUNT (*) AS "Anzahl Lernende" FROM lernende;
```

```
SELECT l.Vorname, l.Nachname, l.Strasse, o.PLZ, o.Ort
  FROM Lernende AS l
  INNER JOIN Orte AS o
  ON l.fk_oid = o.oid;
```

```
SELECT l.Vorname, l.Nachname, l.Strasse, o.PLZ, o.Ort, f.Firma
  FROM Lernende AS l
  INNER JOIN Orte AS o
  ON l.fk_oid = o.oid
  INNER JOIN Firmen AS f
  ON l.fk_fid = f.fid;
```

```
SELECT l.Vorname, l.Nachname, l.Strasse, f.Firma
  FROM Lernende AS l
  LEFT JOIN Firmen AS f
  ON l.fk_fid = f.fid
  ORDER BY l.Vorname;
```

Die **nullte Normalform** ist dann gegeben, wenn alle Informationen in einer Tabelle vorhanden sind und noch **unnormalisiert** vorliegen. Die Nullte Normalform liegt in vielen Fällen während der Anforderungsanalyse einer Datenbank vor. Die Anforderungsanalyse in der Datenbankentwicklung beginnt mit dem Sammeln von unstrukturierten und unsortierten Informationen aus den verschiedenen Fachbereichen.

Rechnungsdaten

RNr	Datum	Kunde	KNr	Adresse	PNr	Artikel	ANr	Anzahl	Preis
123	29.01.2020	Marc Muster	11	Talweg 11, 3000 Bern	1, 2, 3	Monitor, Maus, Bürostuhl	2-0023, 4-0023, 5-0023	10, 12, 1	200 CHF. 25 CHF, 450 CHF
124	30.01.2020	Erika Muster	12	Talweg 21, 3000 Bern	1, 2	Notebook, Maus	1-0023, 1-0023	2, 2	1200 CHF, 25 CHF

Die **erste Normalform 1NF** ist dann gegeben, wenn alle Informationen in einer Tabelle atomar vorliegen. Es bedeutet, dass jede Information innerhalb einer Tabelle eine eigene Spalte bekommt und zusammenhängende Informationen, wie zum Beispiel die Postleitzahl und der Ort, nicht in einer Tabellenspalte vorliegen dürfen. Es bedeutet weiter, dass jedes Objekt bzw. jedes Ereignis innerhalb einer Tabelle eine eigene Zeile bekommt, also zu einem eigenen Datensatz wird. Die einzelnen Rechnungspositionen dürfen nicht zusammengefasst, sondern müssen in eigene Datensätze aufgeteilt werden.

Informationen, die vorher unstrukturiert und unsortiert vorlagen, werden nun einheitlich und klar strukturiert.

RNr	Datum	Vor-name	Nach-name	KNr	Strasse	Haus-nr	PLZ	Ort	PNr	Artikel	ANr	Anz	Preis CHF
123	29.01.2020	Marc	Muster	11	Talweg	11	3000	Bern	1	Monitor	2-023	10	200
123	29.01.2020	Marc	Muster	11	Talweg	11	3000	Bern	2	Maus	4-023	12	25
123	29.01.2020	Marc	Muster	11	Talweg	11	3000	Bern	3	Bürostuhl	5-023	1	450
124	30.01.2020	Erika	Muster	12	Bergweg	21	3000	Bern	1	Notebook	1-023	2	1200
124	30.01.2020	Erika	Muster	12	Bergweg	21	3000	Bern	2	Maus	4-023	2	25

Die **zweite Normalform** ist ein wichtiger Schritt zu einer voll normalisierten relationalen Datenbank. Sie prüft, ob eine **vollständige funktionale** oder nur eine **funktionale Abhängigkeit** von Werten zu einer bestimmten Teilmenge existiert.

Die zweite Normalform wird meistens schon indirekt erreicht, wenn der Datenbankentwickler mit der Erstellung eines ER Diagramms beschäftigt ist. Die logische Aufspaltung von komplexen Sachverhalten zwingt den Datenbankentwickler, Geschäftsprozesse in Relationen abzubilden.

Gute Datenbankentwickler brauchen für die zweite Normalform kein Modell auf dem Papier, sondern können Geschäftsprozesse mit dem Kunden besprechen und direkt in einer Datenbankapplikation implementieren.

Die **dritte Normalform** ist das Ziel einer erfolgreichen Normalisierung in einem relationalen Datenbankmodell. Sie verhindert einerseits Anomalien und Redundanzen in Datensätzen und andererseits bietet sie genügend Performance für SQL-Abfragen.

Die dritte Normalform ist oft ausreichend, um die perfekte Balance aus Redundanz, Performance und Flexibilität für eine Datenbank zu gewährleisten.

RNr	PNr	Artikel	ANr	Anz	Preis CHF
123	1	Monitor	2-023	10	200
123	2	Maus	4-023	12	25
123	3	Bürostuhl	5-023	1	473.50
124	1	Notebook	1-023	2	1200
124	2	Maus	4-023	2	25