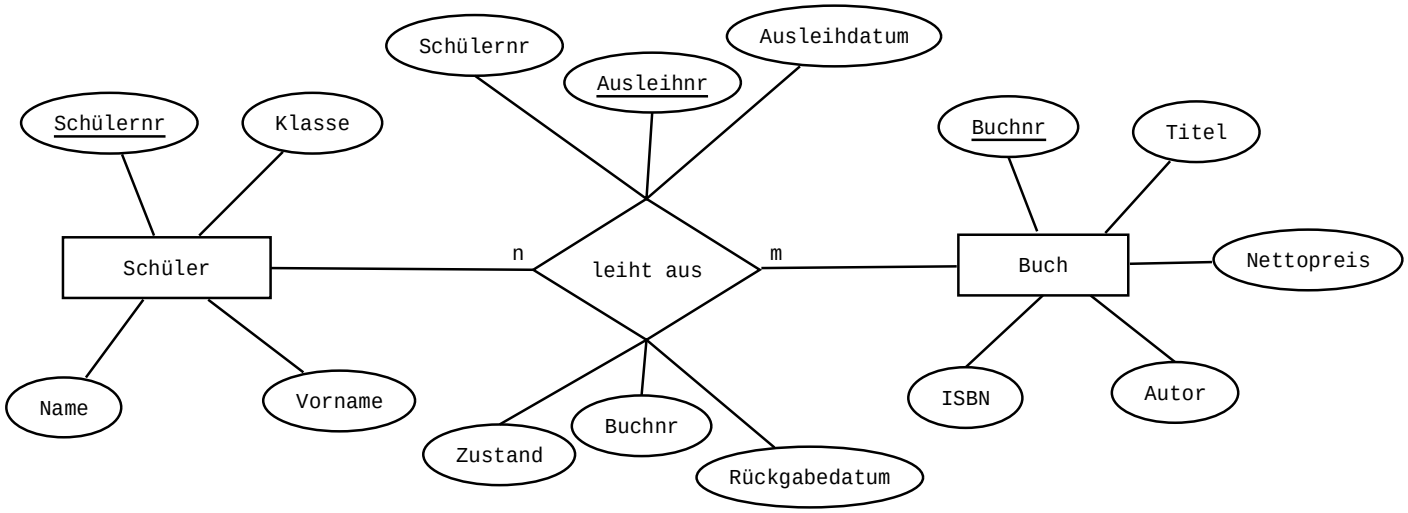
SQL-Abfragen

Nachdem die Datenbank erstellt und mit Daten gefüllt ist, möchten wir mit den Daten arbeiten. Dazu hat sich bei relationalen Datenbanken die Sprache SQL (**S**tructured **Q**uery **L**anguage) durchgesetzt. SQL kann man auch zur Erstellung der Datenbank und zur Manipulation der Daten verwenden. Wir werden uns aber auf den Abfragebereich konzentrieren und Änderungen mit der grafischen Benutzeroberfläche von Open Office/Libre Office durchführen. In der Praxis erstellt man häufig eigene Programme, die Abfragen vereinfachen und Ergebnisse benutzerfreundlicher veranschaulichen.

Um die SQL-Anweisungen zu veranschaulichen, verwenden wir eine Datenbank für eine Schülerbibliothek:



Es sind also drei Tabellen Schüler, Ausleihe, Buch mit den oben dargestellten Attributen (Feldern bzw. Spalten) vorhanden. Natürlich ist dieses ER-Diagramm für eine reale Anwendung unvollständig, aber es reicht um die SQL-Anweisungen zu verdeutlichen.

Einfache Abfragen:

Alle Abfragen beginnen mit `SELECT`, wie z. B.
`SELECT` `*` **`FROM`** `Schüler` oder **`SELECT`** `Titel, ISBN, Autor` **`FROM`** `Buch`
oder **`SELECT`** `*` **`FROM`** `Schüler ORDER BY NAME DESC`

Allgemein:

- **`SELECT`** `Auswahl` **`FROM`** `Tabelle`
  Bei `Auswahl` stehen die anzuzeigenden Felder oder „`*`“, wenn man alle Felder anzeigen lassen möchte. An der Position von `Tabelle` steht der Name der gewünschten Tabelle, z. B. Schüler oder Buch.
- **`SELECT`** `Auswahl` **`FROM`** `Tabelle` **`ORDER BY`** `Feld1 [`**`DESC`**`] [, Feld2 [`**`DESC`**`]]`
  Wenn man die ausgegebenen Daten nach einem Feld oder mehreren sortieren möchte, führt man diese nach ORDER BY auf. Bei absteigender Sortierung (also von Z nach A) fügt man noch DESC hinzu, bei aufsteigender Sortierung (also von

A nach Z) schreibt man nichts hinter dem Feldnamen oder ASC.
Die eckigen Klammern stehen für nicht unbedingt notwendige Elemente.

Auswahl von Datensätzen

Allgemein: `SELECT Auswahl FROM Tabelle WHERE Bedingung [ORDER BY Feld1 [DESC]…]`
Neu ist der Teil mit `WHERE Bedingung`. Die Bedingung vergleicht in der Regel zwei Werte, z. B. „Schülernr = 10“. Als Vergleichsoperatoren stehen zur Verfügung: „=“ (gleich), „<>“ (ungleich), „<“ (kleiner als), „>“ (größer als), „<=“ (kleiner oder gleich) und „>=“ (größer oder gleich) oder „BETWEEN … AND …“. Zeichenketten müssen in Anführungszeichen stehen, also z. B. Name = „Müller“.

Wenn man bei einer Zeichenkette nicht den genauen Wert vergleichen möchte, verwendet man als Bedingung `Feld LIKE Muster`. In Muster steht „%“ für beliebig viele Zeichen und „_“ (Unterstrich) für ein beliebiges Zeichen. Beispiele:
`SELECT * FROM Schüler WHERE Name LIKE „M%“` liefert alle Schüler, deren Namen mit M anfangen.
`SELECT * FROM Schüler WHERE Name LIKE „Ma_er“` liefert alle Schüler, deren Namen mit Ma anfangen und mit er enden, also z. B. Maier oder Mayer.

Wenn mehrere Bedingungen alle gleichzeitig zutreffen sollen, muss man die Bedingungen notieren und dazwischen „AND“ setzen. Wenn von mehreren Bedingungen mindestens eine zutreffen soll, muss zwischen ihnen „OR“ stehen. Wenn man das Gegenteil einer Bedingung möchte, verwendet man „NOT(Bedingung)“. Beispiele:

`SELECT * FROM Schüler WHERE NAME LIKE „M%“ AND Klasse LIKE „9%“` liefert alle Schüler aus den 9. Klassen, deren Namen mit M beginnt.

`SELECT * FROM Schüler WHERE NAME LIKE „M%“ OR Klasse LIKE „9%“` liefert alle Schüler aus den 9. Klassen und alle Schüler, deren Namen mit M beginnt.

Abfragen aus zwei Tabellen (Joins)

Wenn man z.B. alle Schüler wissen möchte, die das Buch mit der Buchnummer 10 ausgeliehen haben, benötigt man Informationen aus zwei Tabellen (Schüler und Ausleihe). Eine SQL-Abfrage dazu sieht folgendermaßen aus:

```
SELECT Schüler.Name, Schüler.Vorname, Klasse FROM Schüler INNER JOIN Ausleihe ON Schüler.Schülernr  = Ausleihe.Schülernr
WHERE Buchnr = 10
```

Die Verknüpfung der beiden Tabellen erfolgt durch die Bedingung hinter ON. Die Schülernummer ist ein Fremdschlüssel in der Tabelle Ausleihe und ein Primärschlüssel in der Tabelle Schüler. In einer relationalen Datenbank werden verschiedene Tabelle in der Regel durch Fremd- und Primärschlüssel. Diese nutzt man bei Joins aus.

Wenn ein Feldname in beiden Tabellen vorkommt oder man verdeutlichen möchte, dass das Feld aus einer Tabelle kommt, schreibt Tabellenname.Feldname, wie im Beispiel bei Schüler.Name geschehen. Wenn es zu keinen Missverständnissen kommen kann, darf man den Tabellennamen auch weglassen, wie bei Klasse oder Buchnummer geschehen.

Es gibt noch weitere Arten von Joins, auf die hier nicht eingegangen wird.

Rechnen und Zählen mit SQL

Man kann mit SQL rechnen. Sind z. B. in Bücher-Tabelle die Netto-Einkaufspreise mitangegeben, kann man sich den Preis mit Mehrwertsteuer berechnen lassen:

```
SELECT Name, Nettopreis * 1.19 AS Bruttopreis FROM Bücher
ORDER BY Bruttopreis
```

Im Beispiel kann man erkennen, dass man Nettopreis mit 1,19 (also mit dem aktuellen Mehrwertsteuersatz) multipliziert. Das berechnete Feld erhält mittels „AS“ gleichzeitig einen neuen Feldnamen, einen Alias. Diesen kann man dann auch in der Bedingung bei WHERE oder bei ORDER BY verwenden.

Möchte man die Anzahl der Bibliotheksnutzer aus einer Klasse, z. B. der 10c, bestimmen, braucht man nicht per Hand zu zählen, sondern formuliert folgende Abfrage:

SELECT COUNT(Name) **FROM** Schüler **WHERE** Klasse=”10a”

Neu ist COUNT(Name). Stattdessen hätte man auch COUNT(*) verwenden können, aber COUNT(Name) ist schneller. Außerdem zählt COUNT(Name) nur die Datensätze, die im Feld Name einen Eintrag besitzen und nicht NULL sind. COUNT(…) ist in SQL eine eingebaute Funktion, die die Anzahl der gefundenen Datensätze zählt.

Mittels `SELECT SUM(Nettopreis) FROM Bücher WHERE Autor="Schiller"` berechnet man die Summe der Netto-Einkaufspreise aller Bücher von Schiller in der Bibliothek.

Weitere eingebaute Funktionen sind u.a.: „MAX(…)“ - maximaler Wert, „MIN(…)“ - mininaler Wert, „AVG(…)“ – Mittelwert eines Feldes.

Achtung:
- Die Funktionen dürfen nicht in einer Bedingung hinter WHERE stehen.
- Die Funktionen dürfen nicht zusammen mit normalen Feldern in der Auswahl hinter SELECT stehen, außer man nutzt Gruppierungen (siehe unten).

Gruppieren

Eine Fragestellung mit der Datenbank der Schülerbibliothek könnte jetzt noch sein, dass man den durchschnittlichen Buchpreis für jeden Autor bestimmen möchte:

```
SELECT Autor, AVG(Nettopreis) FROM Bücher
```

Leider funktioniert nicht diese Abfrage nicht. Das Datenbankmanagementsystem kann nicht entscheiden, was es tun soll: Die Autoren ausgeben oder den mittleren Nettopreis der Bücher berechnen. Aber wir können es unterstützen, indem wir mitteilen, dass wir erst alle Bücher nach Autor gruppieren möchten und dann von jeder Gruppe den mittleren Nettopreis berechnen:

```
SELECT Autor, AVG(Nettopreis) FROM Bücher GROUP BY Autor
```

Sortieren kann man auch bei gruppierten Abfragen:

```
SELECT Autor, AVG(Nettopreis) AS MittlererPreis FROM Bücher
GROUP BY Autor ORDER BY MittlererPreis
```