Modul 105: Datenbanken mit SQL bearbeiten

SQL Crashkurs I: SELECT-Befehl und WHERE-Klausel

Ausgangslage

Nachdem wir uns mit der Arbeitsumgebung vertraut gemacht haben, wollen wir mit der praktischen Arbeit an Datenbanken loslegen. In SQL gibt es einerseits die Data Definition Language DDL um Datenbanken, Tabellen und Indices anzulegen, zu ändern und zu löschen. Diese Befehle werden wir in diesem Modul nur am Rande anschauen.

Andererseits gibt es die Data Manipulation Language DML, mit der Daten eingefügt, geändert und gelöscht und Abfragen durchgeführt werden. Wir werden das Feld sozusagen von hinten aufrollen und in einem ersten Schritt mit einer Datenbank arbeiten, wo bereits alle Daten vorhanden sind. D.h. wir starten mit Abfragen und schauen den SELECT-Befehl mit allen Klauseln genau an und machen dazu viele praktische Beispiele.

Ziele

- Einführung in SQL
- SELECT-Befehl in Theorie und Praxis
- WHERE-Klausel in Theorie und Praxis
- Übungen: Eigene Abfragen formulieren (nur eine Tabelle)

Vorgehen

Eine Übungsdatenbank liegt vor, die Tabellen sind erstellt und die Daten vorhanden. Kopieren Sie die Datenbank *schuelerV1.sqlite* vom Moodle ins Verzeichnis Dokumente/M105/AB105-02 auf der *bmLP1*. Starten Sie darauf *Sqliteman* und öffnen Sie die Datenbank, um mit dem Arbeitsblatt loslegen zu können.

Gehen Sie dieses Arbeitsblatt Schritt für Schritt durch. Konsultieren Sie die Unterlagen, suchen Sie zusätzliche Informationen im Internet oder fragen Sie Ihre Lehrperson.

Unterlagen / Quellen

- SQL Referenz auf Moodle: SQL-Referenz.pdf
- Allgemeine SQL-Tutorials:
  - https://de.wikibooks.org/wiki/Einf%C3%BChrung_in_SQL
  - https://www.w3schools.com/sql
- SQLite-spezifisches Tutorial:
  - https://www.tutorialspoint.com/sqlite/

Zeit: 2 Lektionen

Einführung in SQL

SQL = **S**tructured **Q**uery **L**anguage

SQL ist eine Datenbanksprache zur Definition, Abfrage und Manipulation von Daten in relationalen Datenbanken. SQL ist von ANSI und ISO standardisiert und wird von fast allen gängigen Datenbanksystemen unterstützt. Die beiden wichtigsten Teile von SQL sind die Data Definition Language DDL und die Data Manipulation Language DML.

Die Syntax von SQL ist relativ einfach aufgebaut und semantisch an die englische Umgangssprache angelehnt. SQL stellt eine Reihe von Befehlen zur Verfügung, um folgende Aufgaben zu erledigen:

- Definieren von Datenstrukturen nach der relationalen Algebra. Das wird mit der Data Definition Language DDL erledigt.
- Manipulieren von Datenbestände (Einfügen, Bearbeiten und Löschen). Um diese Aufgaben zu erledigen, steht die Data Manipulation Language DML zur Verfügung.
- Das Abfragen von Daten gehört ebenfalls in den Bereich der DML.

Eine Relationale Datenbank lässt sich vollständig mit SQL erstellen und bewirtschaften. Datenbankspezialisten setzen hauptsächlich SQL ein. Was ist die Alternative? Als Alternative gibt es grafische Tools, mit denen sich gewisse Aufgaben leichter erledigen lassen (wie z.B. *Sqliteman*). Jedoch kommt man trotz grafischen Tools nicht darum herum, immer wieder SQL einzusetzen.

Welche Jobs in der IT-Branche erfordern den Einsatz von SQL? Einerseits arbeiten Applikationsentwickler häufig mit Datenbanken. Oft steht oberhalb von SQL eine Schnittstelle zur Verfügung, die zur Datenverarbeitung verwendet werden kann. Es kann aber sein, dass es keine solche Schnittstelle gibt und Sie mit SQL arbeiten müssen. Oder der Auftraggeber wünscht eine Ad-Hoc-Abfrage, die in der Anwendung nicht vorgesehen ist.

Andererseits arbeiten auch Systemtechniker mit SQL; Es kann zu deren Aufgabengebiet gehören, Datenbanken aufzusetzen und zu betreuen.

Die Wahrscheinlichkeit ist gross, dass Sie irgendwann in Ihrem Berufsleben mit SQL arbeiten werden!

DML – Data Manipulation Language

Befehle:

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

Aggregatsfunktionen:

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

DDL – Data Definition Language

Befehle:

- **`CREATE`** `DATABASE / TABLE / INDEX` = Datenbank / Tabelle / Index erstellen
- **`ALTER`** `DATABASE /TABLE / INDEX` = Datenbank / Tabelle / Index ändern
- **`DROP`** `DATABASE / TABLE / INDEX` = Datenbank / Tabelle / Index löschen

Tabelle erstellen:
```
CREATE TABLE kontakte
("name" TEXT(25), "vorname" TEXT(25));
```

Spalte hinzufügen:
```
ALTER TABLE kontakte ADD COLUMN "ortnr" INTEGER;
```

(Spalte löschen:
```
ALTER TABLE kontakte DROP COLUMN "ortnr";) in sqliteman
nicht möglich
```

SELECT

Der SELECT-Befehl dient dazu, Datensätze aus einer oder mehreren Tabellen anzuzeigen. Um den Befehl und die Klauseln kennenzulernen, beschränken wir uns in diesem Arbeitsblatt auf eine Tabelle. Tabellenverbunde werden Sie später kennenlernen.

Das Resultat einer SELECT-Abfrage ist die SQL-Ergebnismenge (englisch = SQL result set), diese wird in Form einer Tabelle ausgegeben.

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 ];
```

Mindestens erforderlich sind: Das Schlüsselwort SELECT, eine Liste von Spalten (<select list>), das Schlüsselwort FROM und ein Tabellenname (<table reference list>).

Beispiel: `SELECT Name, Vorname, Ort_P FROM schueler;`
Es werden die Namen und Orte von allen Schülern ausgegeben.

Wir bauen in der Folge auf diesem Beispiel auf!

Case Sensitivity (Unterscheidung Gross- und Kleinbuchstaben)

In SQL wird je nach Fall zwischen Gross- und Kleinbuchstaben unterschieden:

Was	Unterscheidung	Beispiel
Schlüsselwörter	nein	SELECT = select
Tabellennamen	nein	"Schueler" = "schueler" = "SCHUELER"
Attributnamen	nein	"Vorname" = "vorname" = "VORNAME"
WHERE-Bedingung	ja	WHERE Nachname > "D" ≠ WHERE Nachname > "d" (Testen Sie beide Abfragen mit der Tabelle *Schueler*)

Auch wenn Gross- und Kleinschreibung meist keine Rolle spielt, wollen wir uns an folgende – allgemein übliche – Schreibweise halten:

- Schlüsselwörter werden grossgeschrieben
- Attribut- und Tabellennamen werden so geschrieben wie beim Anlegen der Tabellen (bzw. so, wie sie in der Datenbank stehen)
- Beispiel: `SELECT Vorname, Name FROM schueler;`

Beschränkung auf eine Anzahl Zeilen

```
LIMIT <number>
```

Wenn nur eine bestimmte (bzw. maximale) Anzahl von Zeilen ausgegeben werden soll, kann die LIMIT-Klausel ans Ende des SELECT-Befehls gesetzt werden.

```
SELECT Name, Vorname, Ort_P FROM schueler LIMIT 10;
```

➔ Es werden die Namen und Orte der ersten 10 Schüler ausgegeben.

Hinweis: Die LIMIT-Klausel ist SQLite-spezifisch, in andern Datenbanken heisst die Klausel anders.

DISTINCT | ALL

DISTINCT

Mit dem Schlüsselwort DISTINCT erhalten Sie nur eindeutige Ergebnisse. Z.B. gibt es unter den Schülern Vornamen, die mehrfach vorkommen. Mit DISTINCT können diese herausgefiltert werden.

Aufgabe 1

```
SELECT DISTINCT Vorname FROM schueler;
```

➔ Gibt alle Vornamen der Schüler aus, doppelte Namen werden zu einem zusammengefasst.

Wie viele verschiedene Vornamen gibt es?

Aufgabe 2

```
SELECT DISTINCT Vorname, Ort_P FROM schueler;
```

Was ist die Bedeutung dieses SELECT?

Wie viele Datensätze werden ausgegeben?

ALL

ALL ist das Gegenstück zu DISTINCT: Es werden ausdrücklich alle Datensätze zurückgegeben, auch doppelte. ALL ist der Standard und wird im Normalfall weggelassen.

<Spaltenliste>

Die Liste der Spalten ist bei jedem SELECT obligatorisch:

```
SELECT Name, Vorname, Ort_P FROM schueler;
```

➔ Gibt die Attribute *Nachname*, *Vorname* und *Ort_P* der Tabelle *schueler* aus. Die Liste kann alle Attributnamen der Tabelle *schueler* enthalten.

**-Zeichen*

Das Zeichen * hat eine Sonderbedeutung:

```
SELECT * FROM schueler;
```

➔ Gibt alle Attribute der Tabelle *schueler* aus.

AS

Die Spaltentitel für die Ausgabe können angepasst werden:

```
SELECT Name, Vorname, Ort_P AS "Ort" FROM schueler;
```

➔ Gibt die Attribute *Nachname*, *Vorname* und *Ort_P* der Tabelle *schueler* aus, mit folgenden Spaltentiteln: Nachname, Vorname, Ort.

Hinweis: In vielen Fällen funktioniert `AS "Nachname"` auch ohne Hochkomma, also `AS Nachname`. Gewöhnen Sie sich jedoch an, **immer** Hochkommas zu verwenden. In einigen Fällen funktioniert es sonst nicht: Z.B. bei Berechnungen oder wenn ein Leerschlag oder ein anderes Sonderzeichen im Titel steht.

Beispiel eines Titels mit Leerschlägen:

```
SELECT Name, Vorname, Ort_P AS "Ort privat" FROM schueler;
```

Aufgabe 3

Notieren Sie den SELECT-Befehl, um Nachname, Vorname, Adresse privat, PLZ privat und Ort privat von den ersten 100 Schülern auszugeben. Vergeben Sie dabei sinnvolle Spaltentitel (z.B. soll "_P" überall weggelassen werden).

FROM

Wir beschränken uns im Moment auf einzelne Tabellen ohne Verknüpfung, d.h. nach dem Schlüsselwort FROM steht jeweils genau ein Tabellenname.

Aggregatsfunktionen und Berechnungen

Berechnungen

Sie können bei numerischen Attributen Berechnungen im SELECT verwenden:

```
SELECT Artikelname, Preis * 1.2 FROM artikel;
```

➔ In diesem Beispiel wird der Preis von Euro in CHF umgerechnet (diesen SELECT können Sie nicht in der aktuellen Datenbank ausführen!).

Aufgabe 4

Geben Sie Nachname, Vorname und das Alter von allen Schülern aus. Als Spaltentitel soll "Nachname", "Vorname" und "Alter" stehen.

SUM()

Mit SUM() kann eine Summe über eine numerische Spalte gebildet werden:

```
SELECT SUM(Preis) FROM artikel;
```

➔ Es wird die Summe der Preise von allen Artikeln ausgegeben (diesen SELECT können Sie nicht in der aktuellen Datenbank ausführen!).

Aufgabe 5

Geben Sie die Summe des Alters von allen Schülern mit einem entsprechenden Titel aus. Wir machen das zu Übungszwecken, diese Berechnung macht nicht wirklich Sinn (oder man könnte sagen, das sei die Lebenserfahrung von allen Schülern zusammen…).

Wie hoch ist die Summe?

AVG()

Mit AVG() kann der Durchschnitt der Werte einer numerischen Spalte berechnet werden.

Aufgabe 6

Berechnen Sie das Durchschnittsalter von allen Schülern und geben Sie es mit einem entsprechenden Titel aus.

Wie hoch ist das Durchschnittsalter?

Zusatzaufgabe 6a

Runden Sie das Durchschnittsalter auf 2 Stellen nach dem Komma.

COUNT()

Mit COUNT() wird die Anzahl der ausgegebenen Datensätze angezeigt. In der Klammer steht normalerweise *, also COUNT(*) oder es kann irgendein Attributname sein.

Aufgabe 7

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

Wie viele sind es?

Mit COUNT(DISTINCT <Attributname>) wird die Anzahl der ausgegebenen Datensätze angezeigt, wobei identische Attributwerte (Liste der Attribute nach DISTINCT) nur 1x gezählt werden.

Aufgabe 8

Geben Sie die Anzahl der unterschiedlichen Vornamen mit einem entsprechenden Titel aus.

Wie viele sind es?

MIN() und MAX()

Die Funktionen MIN() für Minimum und MAX() für Maximum können nicht nur bei numerischen, sondern auch bei alphanumerischen Attributen verwendet werden.

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

➔ Zeigt den tiefsten und höchsten Jahrgang aller Schüler an.

Aufgabe 9

Geben Sie den ersten und letzten Nachnamen gemäss Alphabet mit entsprechenden Titeln aus.

Welches sind die beiden Namen?

WHERE

Ein SELECT ohne WHERE gibt als Resultat immer die gesamte Tabelle zurück. In vielen Fällen wollen wir jedoch nur eine Teilmenge oder einen einzelnen Datensatz betrachten oder bearbeiten. Mit der WHERE-Klausel können wir Bedingungen formulieren, damit wir nur die gewünschten Datensätze erhalten. Der Ausdruck dafür ist *Filter*, d.h. wir filtern die Tabelle, indem wir nur eine Auswahl von Daten durch das Filter hindurch lassen.

Zur Filterung stehen uns folgende Möglichkeiten zur Verfügung:

- Vergleichsoperatoren: = < > <= >= <>
- Boolsche Operatoren: `NOT, AND, OR`
- `BETWEEN…AND`: Die Ergebnismenge liegt im Bereich ZWISCHEN x UND y
- `LIKE`: Ähnlichkeit mit den Platzhaltern `%` und _ (siehe unten)
- `IS NULL`: Prüfung auf "kein Wert"

Weitere Möglichkeiten werden wir zu einem späteren Zeitpunkt anschauen (z.B. IN und EXISTS).

Sämtliche Operatoren und Bedingungen können sowohl bei alphanumerischen (Zeichenketten) als auch numerischen (Zahlen) Wertebereichen angewendet werden!

= und <>

= bedeutet identisch (bei Strings vom 1. bis zum letzten Zeichen)

```
SELECT * FROM schueler WHERE Ort_P = "Bern";
```

➔ Alle Schüler, die in Bern wohnen, werden ausgegeben.

Aufgabe 10

Geben Sie alle Lernenden mit Jahrgang 2001 aus.

Hinweis: Zeichenketten werden immer mit Hochkomma geschrieben, z.B. "Bern". Bei Zahlen hingegen werden keine Hochkommas gesetzt, z.B. 2001. Es funktioniert zwar auch mit Hochkommas ("2001"), jedoch muss der SQL-Interpreter eine Typenkonvertierung von Zeichenkette zu Zahl (in diesem Fall Integer) durchführen!

<> ist das Gegenteil von = oder mit andern Worten: nicht gleich

```
SELECT * FROM schueler WHERE Ort_P <> "Bern";
```

➔ Alle Schüler, die **nicht** in Bern wohnen, werden ausgegeben.

Aufgabe 11

Geben Sie alle Lernenden aus, die einen kleineren oder grösseren Jahrgang als 2001 haben.

<, >, <= und >=

< bedeutet grösser und kann sowohl bei Zahlen als auch Zeichenketten verwendet werden

```
SELECT * FROM schueler WHERE Jahrgang < 2000;
```

➔ Alle Schüler mit Jahrgang 1999 oder kleiner werden ausgegeben.

```
SELECT * FROM schueler WHERE Jahrgang <= 2000;
```

```
SELECT * FROM schueler WHERE Name > "S";
```

```
SELECT * FROM schueler WHERE Name >= "Sommer";
```

➔ Alle Schüler mit Jahrgang 2000 oder kleiner werden ausgegeben.

➔ Gibt alle Schüler aus, deren Nachname mit "S" oder einem Buchstaben weiter hinten im Alphabet beginnt.

➔ Gibt alle Schüler mit Nachnamen "Sommer" und alle, die im Alphabet weiter hinten kommen, aus.

Hinweis: Wie die alphanumerische Sortierung genau funktioniert, erfahren Sie ganz unten in diesem Dokument.

Aufgabe 12

Geben Sie alle Lernenden aus, die 20-jährig oder älter sind.

Geben Sie alle Lernenden aus, deren Nachname mit "A" beginnt (WHERE mit den hier beschriebenen Operatoren verwenden).

AND und OR

Mit AND können Sie zwei Bedingungen verknüpfen, wobei beide Bedingungen wahr sein müssen, damit die Datensätze im Resultatset enthalten sind.

```
SELECT * FROM schueler WHERE Name > "S" AND Ort_P = "Bern";
```

- Gibt alle Schüler aus, deren Nachname mit "S" oder einem Buchstaben weiter hinten im Alphabet beginnt **und** die in Bern wohnen.

Mit OR können Sie zwei Bedingungen verknüpfen, wobei eine oder beide Bedingungen wahr sein müssen, damit die Datensätze im Resultatset enthalten sind.

```
SELECT * FROM schueler WHERE Name > "S" OR Ort_P = "Bern";
```

- Gibt alle Schüler aus, deren Nachname mit "S" oder einem Buchstaben weiter hinten im Alphabet beginnt oder die in Bern wohnen. Es dürfen auch beide Bedingungen erfüllt sein. (Ein gegenseitiger Ausschluss kann mit XOR erreicht werden.)

Aufgabe 13

Geben Sie alle Lernenden aus, deren Nachname mit "C" beginnt (WHERE mit den hier beschriebenen Operatoren verwenden).

Geben Sie alle Lernenden aus, deren Nachname mit "C" oder "F" beginnt (WHERE mit den hier beschriebenen Operatoren verwenden).

Hinweis: Wenn Sie mehr als zwei Bedingungen mit AND und OR verknüpfen, dann verwenden Sie Klammern, damit klar ist, welche Bedingung zu welchem Operator gehört!

NOT

NOT bedeutet eine Umkehrung der Bedingung(en). Verwenden Sie Klammern um den Ausdruck bzw. die Bedingungen, die Sie negieren wollen.

```
SELECT * FROM schueler WHERE NOT (Jahrgang = 2000);
```

➔ Alle Schüler, die nicht den Jahrgang 2000 haben, werden ausgegeben.

Aufgabe 14

Formulieren Sie diesen SELECT um, ohne NOT zu verwenden. Das Resultat muss natürlich dasselbe sein.

Geben Sie alle Schüler aus, die nicht "Finn" mit Vornamen heissen.

BETWEEN <wert1> AND <wert2>

SQL-Befehle, -Klauseln und -Bedingungen sind oft eng an die (englische) Sprache angelehnt. Das gilt auch für BETWEEN…AND: Es ist identisch mit ">= AND <=".

```
SELECT * FROM schueler WHERE Jahrgang BETWEEN 2000 AND 2002;
```

ist identisch mit

```
SELECT * FROM schueler WHERE Jahrgang >= 2000 AND Jahrgang <= 2002;
```

➔ Alle Schüler, deren Jahrgang zwischen 2000 und 2002 liegt.

Aufgabe 15

Geben Sie alle Schüler aus, deren Nachname mit "D" bis "M" beginnt. Verwenden Sie BETWEEN…AND anstatt >= und <=.

LIKE

Bei Betriebssystemen (z.B. in der Konsole von Linux oder Windows) können die Platzhalter (englisch = Wildcards) "*" für beliebige Zeichen und "?" für ein Zeichen verwendet werden.

In SQL gibt es sie auch, nur handelt es sich um andere Zeichen:

Betriebssystem	SQL	Bedeutung
*	%	Platzhalter für beliebige Zeichen, z.B. "M%" = alles, was mit M beginnt
?	_ (Underscore)	Platzhalter für ein Zeichen, z.B. "Ju_er" = die Namen "Juker" und "Jufer" (und allenfalls weitere Namen)

Platzhalter können ausschliessliche mit der LIKE-Bedingung verwendet werden.

```
SELECT * FROM schueler WHERE Name LIKE "M%";
```

➔ Alle Schüler, deren Nachname mit "M" beginnt, werden ausgegeben.

Aufgabe 16

Geben Sie alle Schüler aus, deren Nachname mit "Bra" beginnt.

Geben Sie alle Schüler mit folgenden Nachnamen aus: "Meier", "Meyer", "Mener", usw. (irgendein Buchstabe zwischen "Me" und "er"). Namen wie "Meister" sollen nicht in der Liste erscheinen.

IS NULL

In SQL ist es wichtig, zwischen "" (eine leere Zeichenkette) und NULL zu unterscheiden.

Eigenschaften von NULL:

- Leerer, unbestimmter Wert.
- Wenn ein Datensatz in eine Tabelle eingefügt wird und beim INSERT nicht alle Attribute angegeben werden, dann befindet sich nach dem Einfügen in den fehlenden Attributen der Wert NULL.
- Um einen bestehenden Attributwert zu entfernen, kann man beim UPDATE den Wert NULL angeben.

Eigenschaften einer leeren Zeichenkette:

- Leerer Wert, aber nicht unbestimmt (wie gesagt eine leere Zeichenkette).
- Wenn Daten aus einer CSV-Datei in die Datenbank importiert werden, dann werden fehlende Attributwerte mit einer leeren Zeichenkette eingefügt.
- Um einen bestehenden Attributwert zu entfernen, kann man beim UPDATE den Wert "" (leere Zeichenkette) angeben.

Bei Abfragen muss immer "IS NULL" verwendet werden, "= NULL" ist falsch!

Aufgabe 17

Führen Sie die folgenden zwei Abfragen durch:

```
SELECT * FROM schueler WHERE Firma IS NULL;
```

Wie ist das Ergebnis? Warum?

```
SELECT * FROM schueler WHERE Firma = "";
```

Wie ist das Ergebnis? Warum?

Korrekte Abfrage auf leere Werte

Da wir in der Regel nicht wissen, ob in den Daten leere Zeichenketten oder NULL-Werte vorhanden sind, müssen wir Abfragen immer wie folgt formulieren:

```
SELECT * FROM schueler WHERE Firma = "" OR Firma IS NULL;
```

GROUP BY

Wenn wir alle Wohnorte der Schüler ausgeben wollen, können wir folgende Abfrage verwenden:

```
SELECT DISTINCT Ort_P FROM schueler;
```

Eine zweite Möglichkeit, die dasselbe Ergebnis liefert, ist folgende Abfrage:

```
SELECT Ort_P FROM Schueler GROUP BY Ort_P;
```

Was ist der Sinn von GROUP BY? Diese Klausel wird v.a. dafür verwendet, gruppenweise Auswertungen durchzuführen, wenn im SELECT-Befehl also eine Spaltenfunktion enthalten ist.

```
SELECT Ort_P AS "Ort", AVG(2017-Jahrgang) AS "Durchschnittsalter"
FROM schueler GROUP BY Ort_P;
```

➔ Das durchschnittliche Alter der Schüler pro Ort wird ausgegeben.

Achtung: Bei dieser Abfrage dürfen Sie in der Attributliste nur die Attribute aufführen, die unter GROUP BY aufgeführt sind plus Aggregatsfunktionen. Warum kann z.B. der Name nicht angezeigt werden? Ganz einfach: Es wird nach Ortschaften gruppiert. In Bern wohnen 25 Schüler, "Bern" wird nur 1x angezeigt, welcher Name soll also angezeigt werden?

Aufgabe 18

Zeigen Sie mit einer Abfrage alle Orte an inklusive der Anzahl Schüler, die da wohnen. Natürlich soll jeder Ort nur 1x aufgeführt werden.

HAVING

Mit HAVING kann bei Gruppierungen mit GROUP BY ein Filter gesetzt werden, analog zu den WHERE-Bedingungen.

```
SELECT Ort_P AS "Ort", AVG(2019-Jahrgang) AS "Durchschnittsalter"
FROM schueler GROUP BY Ort_P HAVING Name < "N";
```

➔ Das durchschnittliche Alter der Schüler pro Ort wird ausgegeben. Es werden nur Schüler berücksichtigen, deren Name in der ersten Hälfte des Alphabets liegt.

Aufgabe 19

Zeigen Sie mit einer Abfrage alle Orte an inklusive der Anzahl Schüler, die da wohnen. Es sollen nur die Orte aufgeführt werden, wo mehr als ein Schüler wohnt.

UNION

Mit UNION können ähnliche Tabellen oder Abfragen in einer Ausgabe zusammengefasst werden. Wir werden UNION in diesem Modul nur noch einmal verwenden, im nächsten Arbeitsblatt beim Verbund von Tabellen (für die Realisierung eines FULL OUTER JOIN).

ORDER BY

Mit der ORDER BY-Klausel kann die Ausgabe sortiert werden, was in vielen Fällen Sinn macht.

```
SELECT * FROM schueler ORDER BY Ort_P, Name;
```

➔ Es werden alle Schüler ausgegeben, und zwar sortiert: In erster Linie nach Ort und in zweiter Linie nach Nachname (in den Fällen, wo es mehrere Schüler pro Ort gibt).

ASC

ASC = Ascending oder aufsteigend, d.h. mit dem kleinsten Wert beginnend. ASC ist der Standard und wird im Normalfall weggelassen.

```
SELECT * FROM schueler ORDER BY Ort_P ASC, Name ASC;
```

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

DESC

DESC = Descending oder absteigend, d.h. mit dem grössten Wert beginnend. DESC muss angegeben werden, weil ASC der Standard ist.

```
SELECT * FROM schueler ORDER BY Ort_P DESC;
```

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

Aufgabe 20

Geben Sie alle Schüler aus, und zwar sortiert: In erster Linie nach Ort absteigend und in zweiter Linie nach Nachname absteigend.

Zeigen Sie mit einer Abfrage alle Orte an inklusive der Anzahl Schüler, die da wohnen. Die Liste soll wie folgt sortiert werden: In 1. Linie nach Anzahl Schülern pro Ort absteigend, in 2. Linie nach Ort aufsteigend.

Sortierung von Zeichenketten

Es gibt grundsätzlich zwei Sortierungen in SQL: numerische und alphanumerische.

Betrachten wir folgende Zahlen: 1, 2, 3, 4, 5, 10, 20, 30, 100, 200. Diese Zahlen sind numerisch, aufsteigend sortiert. Die kleinste Zahl kommt an erster, die grösste an letzter Stelle.

Die alphanumerische derselben Zahlen sieht wie folgt aus: 1, 10, 100, 2, 20, 200, 3, 30, 4, 5. Korrekterweise müssten die Zahlen in Hochkommas geschrieben werden, also "1", "10", usw. weil es sich um Zeichenketten und nicht um Zahlen handelt.

Betrachten wir die Sortierungen etwas näher. Bei der numerischen Sortierung ist der Fall klar: Die Höhe der Zahl entscheidet über die Position der Ausgabe.

Bei der alphanumerischen Sortierung kommen folgende Regeln zur Anwendung:

- Es wird nach der ASCII-Tabelle sortiert, wo es für jedes Zeichen einen Code gibt. Ziffern kommen vor Buchstaben, Grossbuchstaben kommen vor Kleinbuchstaben.
- Es wird Zeichen für Zeichen sortiert. "10" kommt vor "2", weil zuerst "1" mit "2" verglichen wird und "1" kleiner ist als "2".

- "Kein Zeichen mehr" kommt vor "weitere Zeichen vorhanden". "2" kommt also vor "20", weil zuerst "2" verglichen wird (ist bei beiden Zahlen identisch) und danach "kein Zeichen" mit "0" verglichen wird. Deshalb kommt "Leu" auch vor "Leuenberger".

Diese Regeln sind wichtig, wenn wir bei SQL-Abfragen Bedingungen mit Zeichenketten formulieren und dabei die Operatoren <, >, <=, >= und die Aggregatsfunktionen MIN() und MAX() verwenden.

WHERE Nachname > "B" bedeutet: Der Buchstabe "B" wird ausgeschlossen, aber alle Nachnamen ab dem Buchstaben "B" sind eingeschlossen.

WHERE Nachname < "M" bedeutet: Alle Nachnamen, die mit "A" bis "L" beginnen, sind eingeschlossen.

Wir haben gelernt, dass die Platzhalter "%" und "_" nur mit LIKE verwendet werden können. Deshalb ist folgende Abfrage **falsch**:

```
SELECT * FROM schueler WHERE Name > "M%";
```

Zwar liefert diese Abfrage das gewünschte Resultat (nämlich alle Nachnamen, beginnend ab "M"). Jedoch ist der Ausdruck trotzdem falsch, denn in diesem Fall wird "%" nicht als Platzhalter, sondern als Zeichen betrachtet. Rein zufällig steht in der ASCII-Tabelle "%" vor allen Buchstaben, weshalb das Resultat korrekt ist. Aber kann jemand von sich behaupten, alle ASCII-Codes zu kennen? Die Abfrage `SELECT * FROM schueler WHERE Name <= "M%"` liefert z.B. nicht das gewünschte Resultat (es werden keine Nachnamen, die mit "M" beginnen, zurückgegeben!).

Zusatzaufgaben

Aufgabe 21

Listen Sie alle Firmen der Tabelle *schueler* auf, aufsteigend alphabetisch sortiert.

Was fällt Ihnen bezüglich der Sortierung auf?

Aufgabe 22

Ergänzen Sie die Abfrage mit der Anzahl Schüler pro Firma (mit entsprechendem Titel).

Aufgabe 23

Zeigen Sie alle Firmen an, die mit "Eidg" beginnen, inkl. Anzahl Schüler.

Aufgabe 24

Zeigen Sie alle Firmen an, die mit "Schweiz" oder "Swiss" beginnen, inkl. Anzahl Schüler.

Aufgabe 25

Führen Sie Nachname, Vorname, PLZ und Ort von den Schülern auf, bei denen die Postleitzahl zwischen 3000 und 3999 liegt. Die Liste soll 1. nach Postleitzahl absteigend und 2. nach Nachname aufsteigen sortiert sein.

Aufgabe 26

Wir benutzten bisher die Tabelle *schueler*, es gibt noch die zweite Tabelle *Lektionen* in der Datebank. Die beiden Tabellen sind durch das Attribut *Klasse* verbunden. Die Datenbank ist nicht in einem guten Zustand, es gibt viele Redundanzen und die Tore für Inkonsistenzen sind weit geöffnet. Für die nächsten Arbeitsblätter wollen wir die Datenbank bereinigen, wenn auch nicht perfektionieren.

Entwerfen Sie ein taugliches, konzeptionelles Datenmodell (siehe Arbeitsblatt 6) für diesen Datenbestand. Redundanzen sollen möglichst vermieden werden, mit einer Ausnahme: Die Zimmer lagern wir nicht in eigene Tabellen aus.