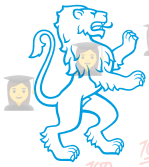


Kantonsschule Im Lee

Informatik

Datenbanken

Skript

Cyril Wendl

Inhaltsverzeichnis

Danksagungen

Besonderer Dank gilt Thomas Graf für seine zahlreichen Beiträge sowie das sorgfältige Lektorat dieses Skripts.

Kapitel 1

Daten & SQL

1.1 Einführung

1.1.1 Was sind Daten?

Der Begriff „Daten“ ist der Plural von *Datum* und kommt aus dem Latenischen (*datum* = gegeben, bzw. *dare* = geben). „Daten“ bedeuten also im weitesten Sinne etwas „**Gegebenes**“, bzw. ein Faktum. Dies wiederum wirft die philosophische Frage auf, was ein Faktum ist. Beispiele von Fakten sind:

- „Mein Nachbar heisst Marco Odermatt.“
- „Aktuell findet an der KLW Unterricht statt.“
- „Das Billett von Zürich nach Winterthur kostet CHF 6.80.“
- „Das Logo der KLW besteht aus blauen und roten Farben.“
- ...

Anhand des letzten Beispiels sieht man, dass Fakten nicht unbedingt wahr sein müssen, um als Fakten zu gelten.

In der **Informatik** sind Daten so gut wie immer durch Nullen und Einsen repräsentiert. Also zum Beispiel:

- ... `10100010101111011101001001` ...
- ... `11101110110000111011010111` ...
- ...

Die Nullen und Einsen können beliebige Informationen darstellen, also beispielsweise Bilder, Texte, Videos oder Zahlen.

1.1.2 Datenmengen

Seit etwa einem guten halben Jahrhundert nehmen die Datenmengen stets zu. Tabelle 1.1 zeigt die gängigen Grössenordnungen von Datenmengen auf.

Masseinheit	**Dezimalsystem**		Grössenordnung
KB (Kilobyte)	10^3 B(yte)	1'000 B	eine Text-Datei
MB (Megabyte)	10^6 B	1'000'000 B	eine Musik-Datei
GB (Gigabyte)	10^9 B	1'000'000'000 B	eine Video-Datei
TB (Terabyte)	10^{12} B	1'000'000'000'000 B	kleiner Firmen-Server
PB (Petabyte)	10^{15} B	...	Facebook-Server
EB (Exabyte)	10^{18} B	...	alle CERN-Daten
ZB (Zettabyte)	10^{21} B	...	alle Daten ($\sim$ 100 ZB)
YB (Yottabyte)	10^{24} B	...	$\sim$ 2030?

Tabelle 1.1: Gängige Daten-Grössenordnungen in der Informatik (ein **B**yte = 8 **b**it)

Die gesamten weltweit vorhandenen Daten machen aktuell ca. 100 Zettabyte aus. Es wird erwartet, dass das erste Yottabyte gegen 2030 erreicht wird (siehe Abbildung 1.1).

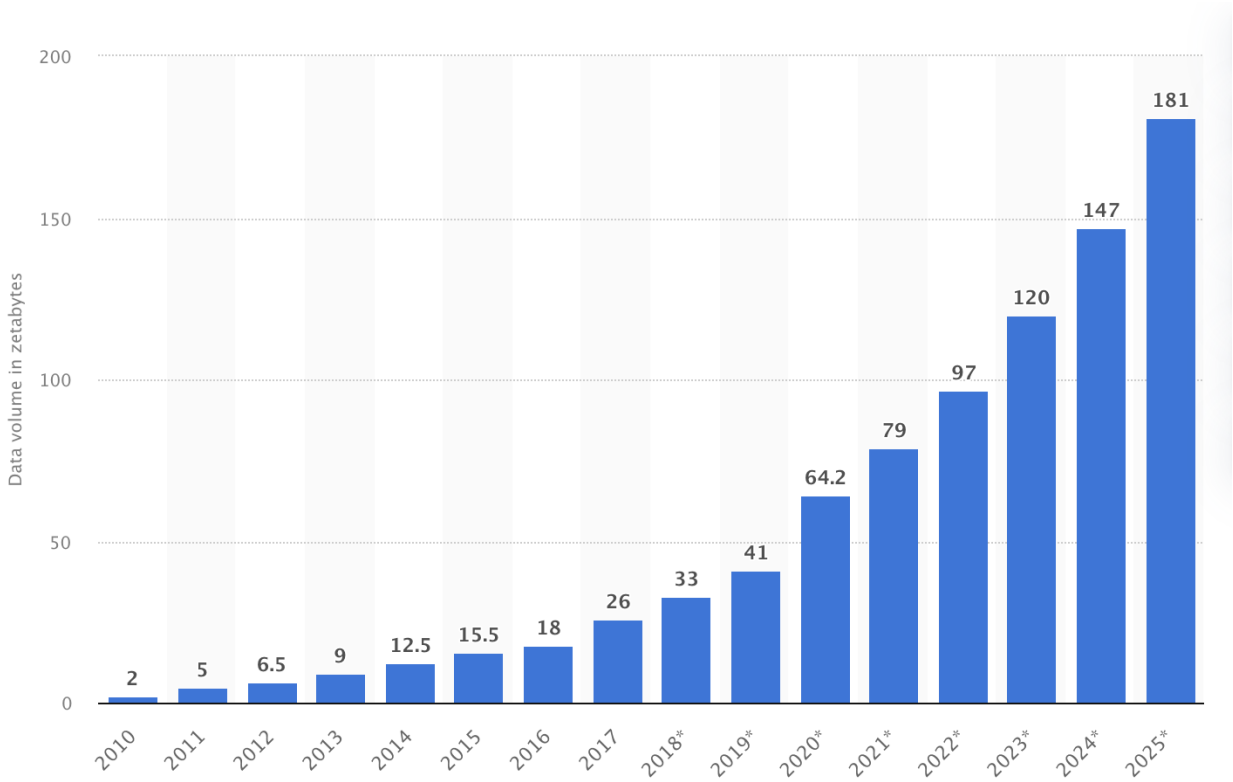


Abbildung 1.1: Entwicklung der globalen Datenmenge

Dies hat zur Folge, dass Datenzentren immer grösser werden und mehr Energie verbrauchen, wie das folgende Video verdeutlicht: `https://youtu.be/_r97qdyQtIk?t=2m14s`

1.1.3 Tabellen

Der allergrösste Teil der oben erwähnten Daten befindet sich in sogenannten Datenbanken. Datenbanken bestehen im Wesentlichen aus Tabellen, weshalb wir im Folgenden zunächst anschauen, wie eine Tabelle aufgebaut ist. Eine Beispiel einer Tabelle ist gegeben in Tabelle 1.2.

Name	Region	Fläche	Einwohner	BIP
Afghanistan	Asien	652	25.8M	21.0B
Albanien	Europa	28.7	3.49M	5.6B
Algerien	Afrika	2,381.7	31.2M	147.6B
Amerikanische...	Ozeanien	0.199	65.4K	0.15B
Andorra	Europa	0.468	66.8K	1.2B
Angola	Afrika	1,246.7	10.1M	11.6B
Anguilla	Mittelamerika	0.091	11.8K	0.088B
Antarktik	Antarktis	14M	0	0
Antigua und...	Mittelamerika	0.442	66.4K	0.524B
Argentinien	Südamerika	2,766.9	36.9M	367B
...	...	...	...	...

Tabelle 1.2: Länder-Tabelle

Eine Tabelle besteht aus Zeilen und Spalten, wie in Tabelle 1.3 gezeigt.

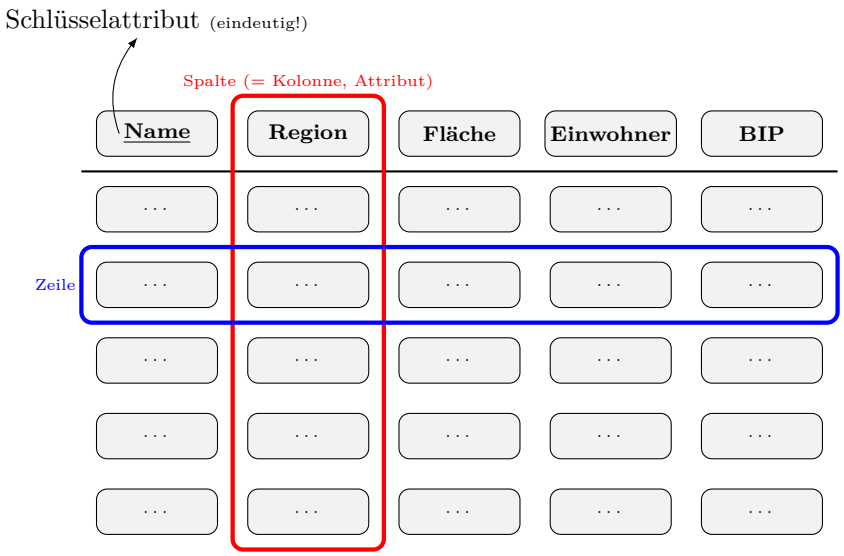


Tabelle 1.3: Typische Tabellenstruktur anhand einer Tabelle mit Informationen zu Ländern

In Tabelle 1.3 enthält jede Zeile Informationen zu einem Land: dessen Name, die Region (=Kontinent), Fläche, die Einwohnerzahl und das Bruttoinlandprodukt (BIP). Jede Spalte enthält Informationen eines selben Typs, beispielsweise die Regionen aller Länder, die Flächen oder die Einwohnerzahlen. In einer Spalte sollten also alle Werte denselben Typ haben, beispielsweise „Zahl“ (Fläche, Einwohner, BIP) oder „Text“ (Name, Region). Nicht zuletzt gibt es in gewissen Tabellen (genau) ein Schlüsselattribut, dessen Werte eindeutig sein müssen. Dies bedeutet in Tabelle 1.3 beispielsweise, dass kein Ländername zweimal oder mehr vorkommen darf. Weshalb dies wichtig ist, werden wir später sehen.

Eine Datenbank ist vereinfacht gesagt eine Sammlung aus mehreren Tabellen, auf die mehrere Geräte zugreifen können (lesen und schreiben), wie in Abbildung 1.2 gezeigt.

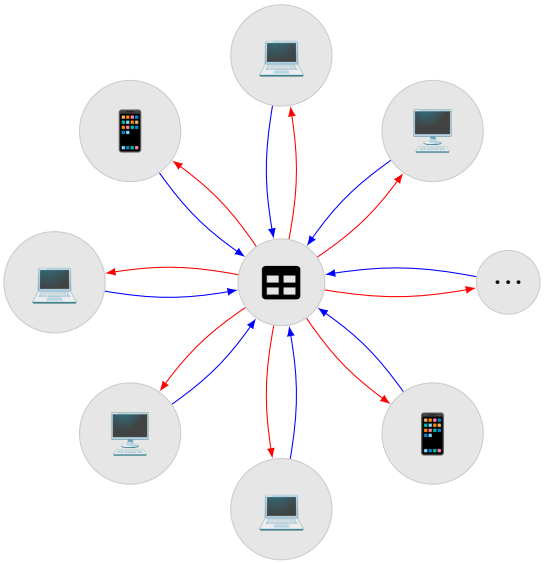


Abbildung 1.2: Geräte können Daten von der / zur Datenbank senden / empfangen

Datenbanken und Tabellen finden Anwendung in allen möglichen Bereichen des täglichen Lebens:

- **Social Media**: Instagram, TikTok, LinkedIn, etc.
- **Shopping**: Galaxus, AliExpress, etc.
- **Netzwerke**: SBB, swissgrid (Strom-Netzwerk), etc.

Wie können wir mit den riesigen Datenmengen umgehen und sinnvolle Einsichten daraus gewinnen?

Eine der verbreitetsten Arten, Tabellen und Datenbanken zu erstellen, zu verändern und zu analysieren ist die Programmiersprache SQL. Im Folgenden verwenden wir die Datenbanken aus der Lern-Plattform InstaHub, um uns mit SQL vertraut zu machen.

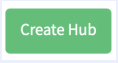
Alle Beispiele können direkt auf InstaHub ausgeführt werden. Die **Abgaben** sind rot markiert und befinden sich auf Moodle unter „Quiz SQL“.

Aufgabe 1.1 Einrichtung von Instahub 📸

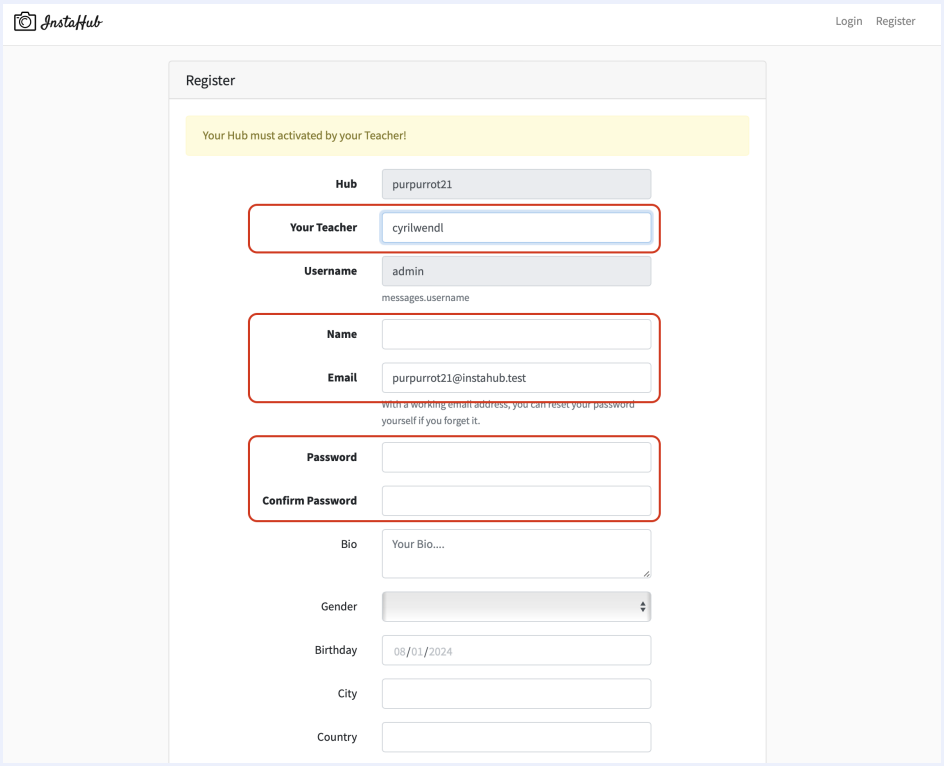
🏁	Selber eine Datenbank administrieren
🧑‍💼	Einzelarbeit
⏱️	ca. 25 min.

Führen Sie folgende Schritte aus:

1. Auf `https://instahub.org` gehen
2. Auf Hub erstellen (rechts oben) klicken und einen Hub erstellen.



3. Geben Sie folgende Angaben ein (siehe Bild unten):
   - Name der Lehrperson (klein geschrieben und ohne Abstände): `cyrilwendl` oder `thomasgraf`
   - Der Username lautet `admin` und kann nicht geändert werden.
   - Ihren Namen und Ihr Passwort können Sie frei setzen. Das Passwort sollten Sie sich irgendwo notieren.
   - Sie können entweder eine echte E-Mail-Adresse angeben oder die generierte E-Mail-Adresse so lassen. Im zweiten Fall können Sie jedoch Ihr Passwort nicht mehr per Mail zurücksetzen, falls Sie dieses je verlieren sollten.
   - Geben Sie mir mündlich Bescheid, dass Sie die Seite erstellt haben. Wir müssen Ihren persönlichen Hub (= Ihre Webseite) manuell aktivieren. Dies dauert nur wenige Sekunden.



4. Merken oder notieren Sie sich Ihr Passwort an einem sicheren Ort.
5. Sie haben nun eine eigene Webseite unter `https://[meinefarbeXX].instahub.org`, wobei `[meinefarbeXX]` durch Ihre Farbe ersetzt werden muss (siehe Bild oben).
6. Sie können sich jetzt auf Ihrer individuellen Webseite mit dem Benutzername `admin` (*nicht* Ihrer Email) und dem von Ihnen zuvor gewählten Passwort anmelden. Sie sind nun als AdministratorIn Ihres eigenen sozialen Netzwerks eingeloggt.

Aufgabe 1.2

Schreiben Sie sich den Namen Ihrer InstaHub-Seite auf, also z.B.

`https://[meinefarbeXX].instahub.org`,

wobei `[meinefarbeXX]` durch Ihre Farbe ersetzt werden muss! Laden Sie diesen auf Moodle unter „Name InstaHub“ hoch, so dass Sie den Link auf Ihr InstaHub später einfach wieder finden. Merken oder notieren Sie sich Ihr Passwort an einem sicheren Ort.

Aufgabe 1.3 Inbetriebnahme von Instahub

🏁	Lernen, mithilfe von SQL eigene Abfragen zu machen
👨‍👩‍👧	2er- bis 3er-Gruppen
⏱️	ca. 60 min.

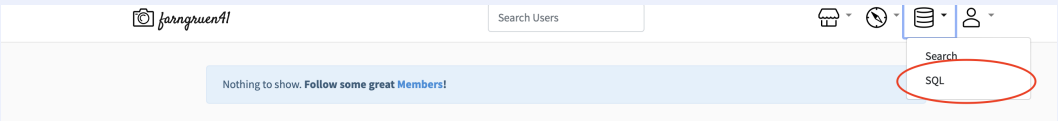
Bestimmen Sie eine Person pro Gruppe, die sich auf der eigenen Webseite mit dem Benutzername `admin` und dem von Ihnen gewählten Passwort anmeldet. Ihre eigene Webseite ist:

`https://meinefarbeXX.instahub.org`

Wobei `[meinefarbeXX]` durch Ihre Farbe ersetzt werden muss.

Die zwei anderen Personen sollen das folgende Kapitel aus diesem Skript lesen und der ersten Person dabei helfen, die SQL-Befehle zu schreiben.

Sie können jetzt unter folgendem Menu-Punkt Ihre eigene Datenbank administrieren:



Als Erstes sehen Sie eine Liste der verfügbaren Tabellen sowie der dazugehörigen Spaltennamen (siehe Bild unten). Tabellennamen sind in Orange, Spaltennamen in grün markiert.

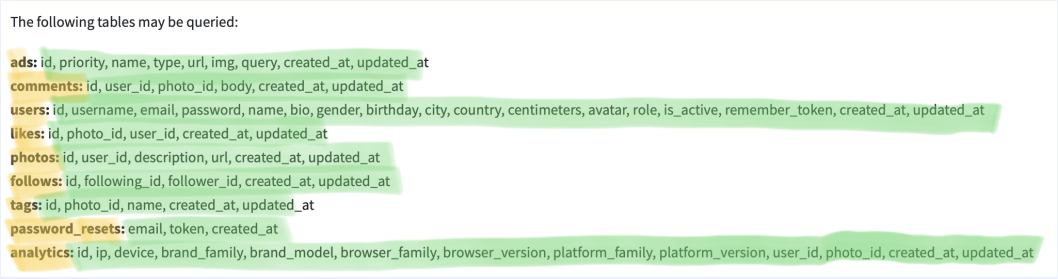


Abbildung 1.3: Die Tabellen von InstaHub und deren Attribute

1.2 Einzelne Tabellen abfragen mit SQL

1.2.1 Syntax

Die Sprache Structured Query Language (SQL) besteht, wie wir sehen werden, aus verschiedenen Schlüsselwörtern, wie etwa `SELECT`, `WHERE`, `GROUP BY`, `ORDER BY` usw. Dazu gibt es Folgendes zu beachten:

- Ob wir diese Schlüsselwörter gross oder klein schreiben, spielt keine Rolle. Allerdings wird empfohlen, die Schlüsselwörter immer in Grossbuchstaben zu schreiben, um sie von Kolonnen-Namen, Tabellen-Namen usw. abzugrenzen.
- Wir können alle SQL-Befehle auf einer einzigen Zeile schreiben, dadurch wird der Code aber deutlich weniger lesbar. Stattdessen empfiehlt es sich, vor gewissen Schlüsselwörtern wie `SELECT`, `WHERE` oder `GROUP BY` eine neue Zeile zu erstellen. Dazu lohnt es sich, die Beispiele genau anzuschauen.

Kommentare können in SQL auf zwei Arten geschrieben werden:

```
-- 1. Kommentar auf einer Zeile
/* 2. Kommentar über
mehrere Zeilen */
```

1.2.2 Einfache Selektion: `SELECT`

Mit dem Befehl „`SELECT col1,col2,... FROM tablename`“ können wir gewisse Spalten (= Kolonnen) `col1,col2,...` aus einer Tabelle mit dem Namen `tablename` auswählen. Wir können statt den Spaltennamen auch einfach einen Stern (`*`) tippen, um alle Kolonnen einer Tabelle zu erhalten, d.h. die gesamte Tabelle.

Beispiel 1.1:
Geben Sie auf InstaHub den folgenden Befehl ein (der Kommentar muss nicht kopiert werden, er dient lediglich zur Erklärung des Codes):

```
-- wähle alle Spalten aus der Tabelle users aus
SELECT *
FROM users
```

Mit diesem Befehl wählen wir alle Spalten (`*`) der Tabelle `users` aus. Sie sollten nun die gesamte Tabelle `users` sehen.

Aufgabe 1.4

Geben Sie alle Benutzernamen (`username`) aus `users` aus.

Lösungsvorschlag zu Aufgabe 1.4

```
SELECT username
FROM users
```

Aufgabe 1.5

Geben Sie die Benutzernamen (`username`) und echten Namen (`name`) aller Einträge aus `users` aus.

Lösungsvorschlag zu Aufgabe 1.5

```
SELECT username, name
FROM users
```

Aufgabe 1.6

Geben Sie die Namen und Wohnorte aller Mitglieder aus.

Lösungsvorschlag zu Aufgabe 1.6

```
SELECT username, city
FROM users
```

1.2.3 Selektion mit Filter: `WHERE`

Fügen wir zu einer SQL-Abfrage den Befehl „`WHERE conditions`“ hinzu, können wir das erhaltene Resultat nach gewissen Bedingungen (`conditions`) filtern.

Beispiel 1.2:
Geben Sie folgendes Beispiel in Instahub ein:

```
SELECT name, city, gender
FROM users
WHERE gender="male"
```

Nebst dem Gleich-Zeichen (im obigen Beispiel) stehen folgende Filter-Operatoren zur Verfügung:

Operator in SQL	Mathematische Bedeutung
`=`	gleich ($=$)
`<>`	ungleich ($\neq$)
`<`	kleiner als ($<$)
`<=`	kleiner oder gleich ($\leq$)
`>`	grösser als ($>$)
`>=`	grösser oder gleich ($\geq$)
`BETWEEN x AND y`	ein Wert zwischen (inklusive) `x` und `y`
`IN (wert1, wert2, ...)`	einer von mehreren möglichen Werten
`IS NULL`	Wert existiert nicht (ist leer)

Um mehrere Bedingungen miteinander zu verknüpfen, können wir die Befehle `AND`, `NOT` sowie `OR` verwenden.

Beispiel 1.3:
Folgender Code gibt alle Mitglieder aus, die zwischen 172 cm und 174 cm gross sind. Dabei gehören 172 cm und 174 cm selbst ebenfalls dazu (inklusive).

```
SELECT name, centimeters, gender
FROM users
WHERE centimeters BETWEEN 172 AND 174
```

Beispiel 1.4:
Folgender Code gibt den Namen und die Stadt aller Mitglieder aus, die entweder in Leipzig, Berlin oder Hamburg wohnen.

```
SELECT name, city
FROM users
WHERE city IN ("Leipzig", "Berlin", "Hamburg")
```

Achtung

Wie Sie anhand der Beispiele 1.3 und 1.4 sehen, spielt es in SQL eine Rolle, ob wir nach Zahlen oder nach Text filtern: Wenn nach einem Text gefiltert wird, muss dieser in Anführungszeichen stehen (wie z.B. `"Berlin"` in Beispiel 1.4). Bei Zahlen dürfen Sie keine Anführungszeichen schreiben (siehe z.B. die Zahl `172` in Beispiel 1.3).

`NULL` ist ein spezielles SQL-Wort, welches bedeutet, dass ein Feld leer ist, bzw. keinen Wert hat. `NULL` muss von „0“ unterschieden werden: Beispielsweise könnte die Zahl 0 in einer Tabelle über Bankkonten bedeuten, dass jemand 0 Franken auf seinem Konto hat, während der Wert `NULL` bedeuten würde, dass keine Informationen zum Kontostand vorhanden sind.

Beispiel 1.5:
Folgender Code gibt die Namen aller Mitglieder an, die ihre Grösse nicht angegeben haben.

```
SELECT name, centimeters
FROM users
WHERE centimeters IS NULL
```

Aufgabe 1.7

Geben Sie die Namen aller weiblichen Mitglieder aus, die zwischen 150 und 155 gross sind.

Lösungsvorschlag zu Aufgabe 1.7

```
SELECT name, gender, centimeters
FROM users
WHERE gender="female"
AND centimeters BETWEEN 150 AND 155
```

Aufgabe 1.8

Zeigen Sie den Namen, das Geburtsdatum sowie die Grössen aller Frauen an, die kleiner oder gleich 160 Zentimeter sind.

Lösungsvorschlag zu Aufgabe 1.8

```
SELECT name, birthday, centimeters
FROM users
WHERE gender="female"
AND centimeters <= 160
```

Aufgabe 1.9

Verwenden Sie die Tabelle `comments` und geben Sie alle Kommentare aus, die vom user mit der ID 10 oder vom user mit der ID 38 stammen.

Lösungsvorschlag zu Aufgabe 1.9

```
SELECT user_id, body
FROM comments
WHERE user_id IN (38, 10)
```

1.2.4 Eindeutige Selektion: `SELECT DISTINCT`

Der Befehl `DISTINCT` nach einem `SELECT` führt dazu, dass jeder mögliche Wert nur höchstens einmal ausgegeben wird, d.h. Duplikate werden entfernt.

Aufgabe 1.10

Vergleichen Sie folgende zwei Befehle. Überlegen Sie sich zuerst, was der Code ausgeben sollte, bevor Sie die Befehle in InstaHub ausführen.

```
SELECT gender
FROM users
```

```
SELECT DISTINCT gender
FROM users
```

Aufgabe 1.11

Geben Sie jeden Wohnort nur einmal aus.

Lösungsvorschlag zu Aufgabe 1.11

```
SELECT DISTINCT city
FROM users
```

Aufgabe 1.12

Geben Sie jede Benutzerrolle nur einmal aus. `dba` bedeutet „Database Administrator“, also Datenbank-AdministratorIn.

Lösungsvorschlag zu Aufgabe 1.12

```
SELECT DISTINCT role
FROM users
```

1.2.5 Ungefähre Treffer: `LIKE`

Mit dem Befehl `LIKE` können wir die Tabelle nach Zeilen filtern, welche in einer gewissen Spalte ein bestimmtes Wort enthalten.

Beispiel 1.6:
Mit folgendem Befehl erhalten wir alle Städte, die mit „Be“ beginnen:

```
SELECT DISTINCT city
FROM users
WHERE city LIKE "Be%"
```

Das „%“-Zeichen ist eine sogenannte *wildcard* und dient hier als Platzhalter, der Folgendes bedeutet: „es *kann* noch weiterer Text an dieser Stelle stehen, *muss* aber nicht“ (siehe Tabelle 1.4).

Symbol	Bedeutung
`"%"`	Stellt null oder mehrere, beliebige Zeichen dar
`"_"`	Stellt ein einzelnes, beliebiges Zeichen dar

Tabelle 1.4: *wildcards* in SQL

Beispiel 1.7:
Mit folgendem Befehl erhalten wir alle Städte, die mit „H“ beginnen und ein „m“ an dritter Stelle haben:

```
SELECT DISTINCT city
FROM users
WHERE city LIKE "H_m%"
```

Aufgabe 1.13

Vergleichen Sie folgende drei Befehle. Überlegen Sie sich zuerst, was der Code ausgeben sollte, bevor sie die Befehle in InstaHub ausführen.

```
SELECT username, city
FROM users
WHERE city = "Berlin" AND name LIKE "Fabian%"
```

```
SELECT username, city
FROM users
WHERE city = "Berlin" OR name LIKE "Fabian%"
```

```
SELECT username, city
FROM users
WHERE city = "Berlin" AND NOT name LIKE "Fabian%"
```

Aufgabe 1.14

Finden Sie alle Berliner, die *Marc* heissen.

Lösungsvorschlag zu Aufgabe 1.14

```
SELECT name, city
FROM users
WHERE city="Berlin" AND name LIKE "Marc%"
```

Aufgabe 1.15

Finden Sie alle Person mit dem Vornamen *Lina* oder *Lorena*.

Lösungsvorschlag zu Aufgabe 1.15

```
SELECT name, city
FROM users
WHERE name LIKE "Lina%"
OR name LIKE "Lorena%"
```

Aufgabe (Challenge) 1.16

Überprüfen Sie, welche Mitglieder im Jahr 2001 geboren sind (mit der Spalte `birthday`).

Lösungsvorschlag zu Aufgabe 1.16

```
SELECT name, birthday
FROM users
WHERE birthday LIKE "2001%"
```

Aufgabe 1.17

Wählen Sie alle Personen mit dem Namen *Naomi* aus, die nicht aus Berlin kommen.

Lösungsvorschlag zu Aufgabe 1.17

```
SELECT name, city
FROM users
WHERE name LIKE "Naomi%"
AND city <> "Berlin"
```

Aufgabe (Challenge) 1.18

Geben Sie die Grösse und den Wohnort von Juliette Amsel sowie Juliette Unger aus. Sie sollten das Zeichen `"%"` nicht verwenden.

Lösungsvorschlag zu Aufgabe 1.18

```
SELECT name, centimeters, city
FROM users
WHERE name LIKE "Juliette ___e_"
```

1.2.6 Sortieren: `ORDER BY`

Mit dem Befehl `ORDER BY col ASC` oder `ORDER BY col DESC` kann das Resultat einer SQL -Abfrage nach einer bestimmten Kolonne `col` sortiert werden. Mit `ASC` wird das Resultat aufsteigend und mit `DESC` absteigend sortiert.

Beispiel 1.8:
Beobachten Sie das Resultat folgender Abfrage:

```
SELECT DISTINCT city
FROM users
ORDER BY city DESC
```

Aufgabe 1.19

Geben Sie alle Benutzernamen in sortierter Reihenfolge aus (a → z).

Lösungsvorschlag zu Aufgabe 1.19

```
SELECT username
FROM users
ORDER BY username ASC
```

Aufgabe 1.20

- Lösen Sie Aufgaben 1–11 unter folgendem Link: https://sql-tutorial.de/home/uebungen.php?lektion=1 (keine Abgabe notwendig / möglich)
- Geben Sie danach auf Moodle die „Übungen 1 (Einfache Selektion)“ ab.

1.2.7 Aggregatsfunktionen

1.2.7.1 `COUNT`

Mit dem Befehl `COUNT` kann man die Anzahl Resultate zählen.

```
SELECT COUNT(*)
FROM users
```

Wir können den berechneten Wert zudem zur besseren Lesbarkeit umbenennen, indem wir den Befehl `AS` verwenden:

```
SELECT COUNT(*) AS "Registrierte Mitglieder"
FROM users
```

Aufgabe 1.21

Geben Sie die Anzahl registrierten Mitglieder in Berlin aus. Die resultierende Spalte soll „Registrierte Mitglieder in Berlin“ heissen.

Lösungsvorschlag zu Aufgabe 1.21

```
SELECT COUNT(*) AS "Registrierte Mitglieder in Berlin"
FROM users
WHERE city="Berlin"
```

1.2.7.2 `MAX`, `MIN`

Um die höchsten, bzw. tiefsten Werte einer Kolonne zu erhalten, können die Befehle `MIN` und `MAX` verwendet werden:

Beispiel 1.9:
Beobachten Sie das Resultat folgender Abfrage:

```
SELECT MAX(centimeters) AS "Maximale Körpergrösse"
FROM users
```

Aufgabe 1.22

Zeigen Sie, wie gross das kleinste Mitglied ist.

Lösungsvorschlag zu Aufgabe 1.22

```
SELECT MIN(centimeters) AS "Kleinste Körpergrösse"
FROM users
```

Aufgabe 1.23

Zeigen Sie, wann sich zuletzt ein Mitglied registriert hat.

Lösungsvorschlag zu Aufgabe 1.23

```
SELECT max(created_at) AS "Zuletzt Registriert"
FROM users
```

1.2.7.3 SUM

Mit SUM kann die Summe einer Datenserie ausgegeben werden.

Beispiel 1.10:
Folgender Code berechnet die Summe der Körpergrössen aller Personen, die Felix heissen:

```
SELECT SUM(centimeters)
FROM users
WHERE name LIKE "Felix%"
```

1.2.7.4 AVG

Mit AVG kann der Durchschnitt einer Datenserie ausgegeben werden.

Beispiel 1.11:
Folgender Code berechnet die durchschnittliche Körpergrössen aller Personen, die Felix heissen:

```
SELECT AVG(centimeters)
FROM users
WHERE name LIKE "Felix%"
```

1.2.7.5 LENGTH

Mit LENGTH gibt die Anzahl Zeichen eines Texts zurück.

Beispiel 1.12:
Folgender Code berechnet die Länge aller Namen:

```sql
SELECT name, LENGTH(name)
FROM users
```

Aufgabe 1.24

- Lösen Sie Aufgaben 1–6 unter diesem folgendem Link (keine Abgabe notwendig / möglich)
- Geben Sie danach auf Moodle die „Übungen 2 (Aggregatsfunktionen)“ ab.

1.2.8 Gruppieren: `GROUP BY`

Um die Resultate einer Anfrage pro Untergruppen zu sehen, kann der Befehl `GROUP BY` verwendet werden.

Beispiel 1.13:
Folgender Code gibt die Anzahl Mitglieder pro Stadt aus:

```sql
SELECT city, COUNT(*) AS "Mitglieder pro Stadt"
FROM users
GROUP BY city
```

Zudem ist es möglich, Daten auch in Untergruppen zusammenzufassen.

Beispiel 1.14:
Folgender Code gibt die Anzahl männlicher und weiblicher Mitglieder pro Stadt aus:

```sql
SELECT city, gender, COUNT(*)
FROM users
GROUP by city, gender
```

1.2.9 Filtern nach Gruppieren: `HAVING`

Falls nach einem `GROUP BY` die Resultate noch weiter eingegrenzt werden sollen, muss statt `WHERE` der Befehl `HAVING` verwendet werden. Der Befehl `WHERE` wird verwendet, um die ursprünglichen Daten *vor* einem `GROUP BY` zu filtern, der Befehl `HAVING` wird verwendet, um die Resultate *nach* einem `GROUP BY` zu filtern.

Aufgabe (Challenge) 1.25

Geben Sie die durchschnittliche Körpergrösse aller Mitglieder in jeder Stadt aus. Zeigen Sie nur die Städte, in denen die Menschen im Durchschnitt zwischen 150 und 155 gross sind.

Lösungsvorschlag zu Aufgabe 1.25

```
SELECT city, AVG(centimeters) AS "Durchschnittliche Körpergrösse"
FROM users
GROUP BY city
HAVING `Durchschnittliche Körpergrösse` BETWEEN 150 AND 155
```

Aufgabe (Challenge) 1.26

Geben Sie die maximale Körpergrösse aus, gruppiert nach Stadt und Geschlecht, für alle Städte, die mit dem Buchstaben ”B”beginnen

Lösungsvorschlag zu Aufgabe 1.26

```
SELECT city, gender, MAX(centimeters) AS "Grösste Körpergrösse"
FROM users
WHERE city LIKE "B%"
GROUP BY city, gender
```

1.2.10 Erste / Letzte Zeilen: `LIMIT`

Mit dem Befehl `LIMIT n` kann das Resultat einer beliebigen SQL-Abfrage auf eine gewisse Anzahl Zeilen beschränkt werden. Dies bedeutet, dass das Resultat nach den ersten `n` Zeilen abgeschnitten wird. Der Befehl steht immer am Ende einer SQL-Abfrage.

Beobachten Sie das Resultat folgender Abfrage:

Aufgabe 1.27

Zeigen Sie nur 3 Mitglieder (nur deren Namen) an.

Lösungsvorschlag zu Aufgabe 1.27

```
SELECT name
FROM users
LIMIT 3
```

Aufgabe 1.28

Geben Sie die drei Städte mit den meisten Mitgliedern an.

Lösungsvorschlag zu Aufgabe 1.28

```
SELECT city, COUNT(*) AS "Anzahl Mitglieder"
FROM users
GROUP BY city
ORDER BY `Anzahl Mitglieder` DESC
LIMIT 3
```

Aufgabe 1.29

Zeigen Sie die Namen und Körpergrösse der 5 grössten Mitglieder an.

Lösungsvorschlag zu Aufgabe 1.29

```
SELECT name, centimeters
FROM users
ORDER BY centimeters DESC
LIMIT 5
```

Aufgabe (Challenge) 1.30

Geben Sie die Stadtnamen aus, wo die meisten Mitglieder mit einem „b“ im Namen wohnen (nur die ersten drei Zeilen).

Tipps:

- Berechnen Sie zuerst mit einem `GROUP BY`-Befehl die Anzahl Einwohner pro Stadt
- Beschränken Sie danach Ihre Abfrage mit dem `WHERE`-Befehl auf Benutzer, die ein „b“ im Namen haben
- Sortieren Sie dann nach der Anzahl Mitglieder
- Verwenden Sie den `LIMIT`-Befehl erst am Schluss

Lösungsvorschlag zu Aufgabe 1.30

```
SELECT city, COUNT(*) AS "Mitglieder"
FROM users
WHERE name LIKE "%b%"
GROUP by city
ORDER BY Mitglieder DESC
LIMIT 3
```

Aufgabe 1.31

- Lösen Sie Aufgaben 1–5 unter diesem Link.
- Geben Sie danach auf Moodle „Übungen 3 (`GROUP BY`)“ ab.

1.2.11 Verzweigungen: `CASE WHEN`

Der SQL-Befehl `CASE WHEN` kann helfen, Ausdrücke basierend auf Bedingungen zu schreiben, also ähnlich wie if-elif-else-Verzweigungen in Python.

Beispiel 1.15:
Die durchschnittliche Körpergrösse in Deutschland beträgt 179 cm für Männer. Folgender Code berechnet für alle Männer, ob sie grösser, kleiner oder gleich dem Durchschnitt sind, und gibt das Resultat in einer neuen Kolonne aus.

```
SELECT name, centimeters,
CASE
    WHEN centimeters > 179 THEN 'Gross'
    WHEN centimeters < 179 THEN 'Klein'
    ELSE 'Genau im Durchschnitt'
END AS Durchschnittlich
FROM users
WHERE gender="male"
```

Aufgabe 1.32

Berechnen Sie eine neue Kolonne, die den Text „langer Name“ enthält, falls ein Name länger als 20 Zeichen lang ist, und ansonsten „kurzer Name“. BenutzerInnen mit langem Namen sollen zuoberst stehen. Zeigen Sie nur die ersten 5 Zeilen.

Lösungsvorschlag zu Aufgabe 1.32

```
SELECT name, LENGTH(name),
CASE
    WHEN LENGTH(name)>20 THEN "Langer Name"
    ELSE "Kurzer Name"
END AS "Langer oder kurzer Name"
FROM users
ORDER BY `Langer oder kurzer Name` DESC
LIMIT 5
```

1.2.12 Anwendung: Personalisierte Werbung auf InstaHub

Auf InstaHub können Werbungen an verschiedene User angepasst werden. Dafür sind auf InstaHub noch einige weitere Tabellen angelegt, deren Entity-Relationship Model (ERM) in Abbildung 1.4 abgebildet ist.

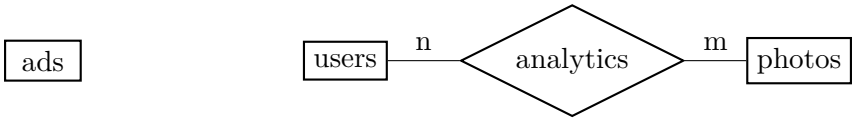


Abbildung 1.4: ERM für den Werbe-Teil von InstaHub

Mit der Tabelle `analytics` beginnt InstaHub das Verhalten der Besucher zu überwachen. Dabei wird der Besuch von Photo-Detailansichten mit folgenden Werten dokumentiert:

Feld	Beschreibung
`id`	Primärschlüssel, fortlaufende Nummer
`ip`	Die ersten drei Blöcke der IPv4-Adresse
`device`	desktop, mobile, tablet oder bot
`brand_family`	wird oft nur bei Smartphones mitgesendet, etwa Apple oder Samsung
`brand_model`	wird oft nur bei Smartphones mitgesendet, etwa GALAXY S5
`browser_family`	Broswer (Firefox, Chrome, Safari...)
`browser_version`	Versionsnummer des Browsers
`platform_family`	Betriebssystem (Windows, Mac, GNU/Linux, iOS, Android...)
`platform_version`	Die Versionsnummer der Plattform
`user_id`	Benutzer, der sich das Foto angesehen hat
`photo_id`	Angesehenes Foto
`created_at`	Zeitpunkt, als das Foto sich angesehen wurde
`updated_at`	i.d.R. wie `created_at`, nur anders, wenn manuell geändert

Tabelle 1.5: Kolonnen der Tabelle `analytics`

Mit diesen Informationen, sowie weiteren Informationen (Hashtags, Geschlecht, Wohnort etc.) kann InstaHub gezielt personalisierte Werbung schalten. Nach diesem Prinzip funktionieren auch andere soziale Medien wie TikTok, Instagram oder Snapchat.

Achtung

Für die nachfolgenden Teile müssen Sie, sofern Sie Adblockers in Ihrem Browser verwenden, diese deaktivieren (oder zumindest eine Ausnahme für InstaHub erstellen), da Ihnen die Werbungen ansonsten nicht angezeigt werden.

Auf InstaHub können Werbeanzeigen mit SQL personalisiert werden. Dafür geht man zunächst auf die Werbungs-Seite (siehe Abbildung 1.5)

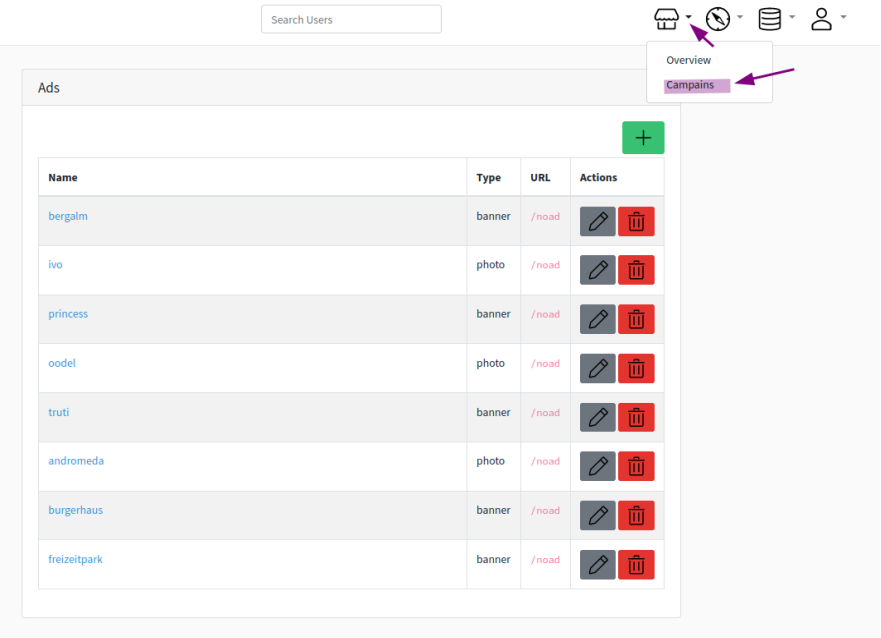


Abbildung 1.5: Übersicht der existierenden Werbungs-Kampagnen

Von dort aus kann man auf eine Werbekampagne klicken, um mehr Details zu sehen. Die Detail-Ansicht der ersten Kampagne ist abgebildet auf Abbildung 1.6.

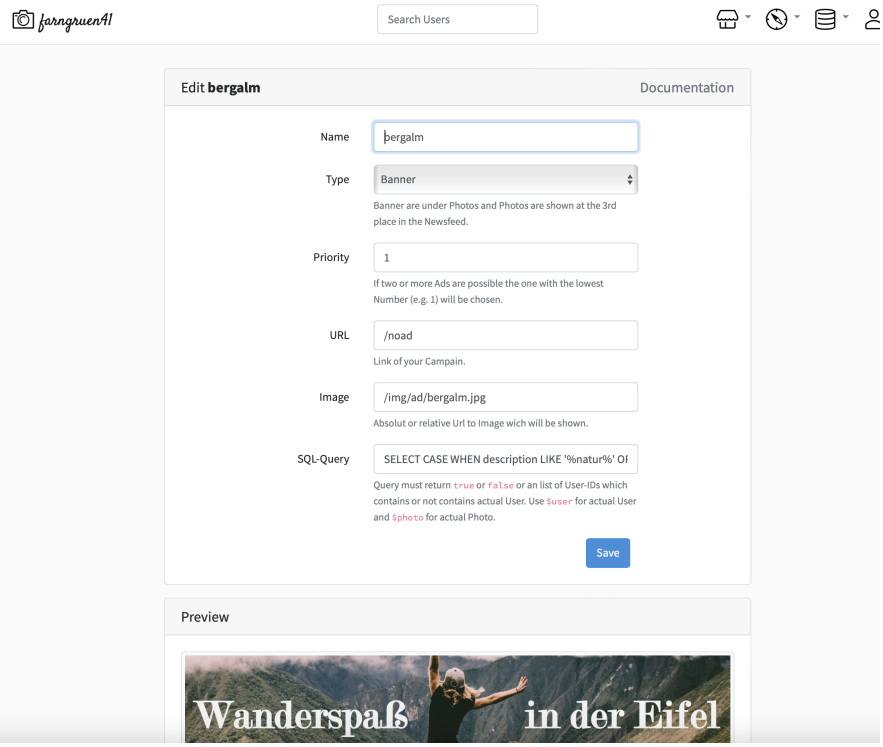


Abbildung 1.6: Beispielkampagne

Als erstes sehen wir den (frei wählbaren) Kampagnen-Namen `bergalm`. Wir sehen auch dass die Werbung vom Typ „banner“ ist, sie wird also bei Fotos angezeigt. Nebst weiteren Informationen sehen wir insbesondere zuunterst die „SQL-Query“, also die SQL-Abfrage, mit der eine Werbung personalisiert wird. Für diese Natur-Werbung haben wir folgende Abfrage:

```
SELECT CASE
    WHEN description LIKE '%natur%'
    OR   description LIKE '%landschaft%'
    OR   description LIKE '%berg%' THEN true
    ELSE false
END
FROM photos
WHERE id=$photo
```

Diese Abfrage wird nun auf der Webseite für jedes Foto ausgeführt, um zu sehen, ob die Werbung zum Foto passt.

Wir sehen zuerst, dass die Kolonne `description` der Tabelle `photos`, also die Kolonne, die Hashtags speichert, auf Texte wie „Natur“, „Landschaft“ und „Berg“ abgesucht wird. Falls so ein Text vorhanden ist in den Hashtags des Fotos, wird der Wert `true` zurückgegeben, und somit wird die Werbung potentiell angezeigt, andernfalls wird der Wert `false` zurückgegeben. Auf der letzten Zeile sehen wir eine Variable `$photo`, mit welcher jedes Foto der gesamten Webseite nach den genannten Kriterien abgesucht werden kann.

Aufgabe 1.33

Verändern Sie den SQL-Befehl so, dass die Werbung auch für Fotos mit dem Hashtag „`#Wasser`“ angezeigt werden. Testen Sie danach, ob es geklappt hat, indem Sie Fotos mit diesem Hashtag suchen. Sie können auf Ihrer Instahub-Webseite überprüfen, ob es geklappt hat:

`[ihrefarbe].instahub.org/p/1517`

Lösungsvorschlag zu Aufgabe 1.33

```
SELECT CASE
    WHEN description LIKE '%natur%'
    OR   description LIKE '%landschaft%'
    OR   description LIKE '%wasser%'
    OR   description LIKE '%berg%' THEN true
    ELSE false
END
FROM photos
WHERE id=$photo
```

Aufgabe (Challenge) 1.34

Erstellen Sie eine neue Werbekampagne (siehe Abbildung 1.5).

Aufgabe (Challenge) 1.35

🏁	Gemeinsam ein soziales Netzwerk verwalten
👨‍👩‍👧‍👦	3er- bis 4er-Gruppen
⏱️	ca. 25 min.

Um sich ein Konto auf einer Seite einer anderen Person zu erstellen:

- Bestimmen Sie jemanden als AdministratorIn und verwenden Sie nur die Webseite dieser Person, also z.B. `https://purpurrot21.instahub.org`. Die anderen Personen sind BenutzerInnen.
- BenutzerInnen: Gehen Sie auf die von der Administratorin erstellte Webseite (`https://namedesnetzwerks.instahub.org`, wobei „`namedesnetzwerks`“ ersetzt werden muss) und erstellen Sie ein neues Konto für sich.
- AdministratorIn: Die neuen Accounts der BenutzerInnen muss dann von Ihnen als AdministratorIn aktiviert werden, indem Sie den User suchen, auf „Bearbeiten“ und dann ganz unten auf „Account ist freigeschaltet“ klicken.
- AdministratorIn und BenutzerIn: Posten Sie nun falls Sie möchten Bilder in den sozialen Netzwerken Ihrer Kollegen, vergeben Sie Likes, folgen Sie anderen und kommentieren Sie fleissig in den anderen Netzwerken (max. 5 Minuten).
- Erstellen Sie nun personalisierte Werbungen für Ihre KlassenkameradInnen (siehe Abbildung 1.5).

1.3 ERM

Ein Entity-Relationship Model (ERM) ist eine abstrakte Darstellung der Tabellen (Entities), die in einer Datenbank angelegt sind, sowie der Beziehungen (Relationships) zwischen den Tabellen. Jede Tabelle kann mehrere Kolonnen (=Attribute) haben.

Wenn man eine Datenbank erstellt, kann es nützlich sein, zuerst ein ERM aufzuzeichnen, um sich zu überlegen, welche Tabellen es braucht.

Das ERM des sozialen Netzwerks InstaHub ist in Abbildung 1.7 abgebildet. Abbildung 1.8 zeigt die Darstellung der Komponenten eines ERM in der sogenannten Chen-Notation.

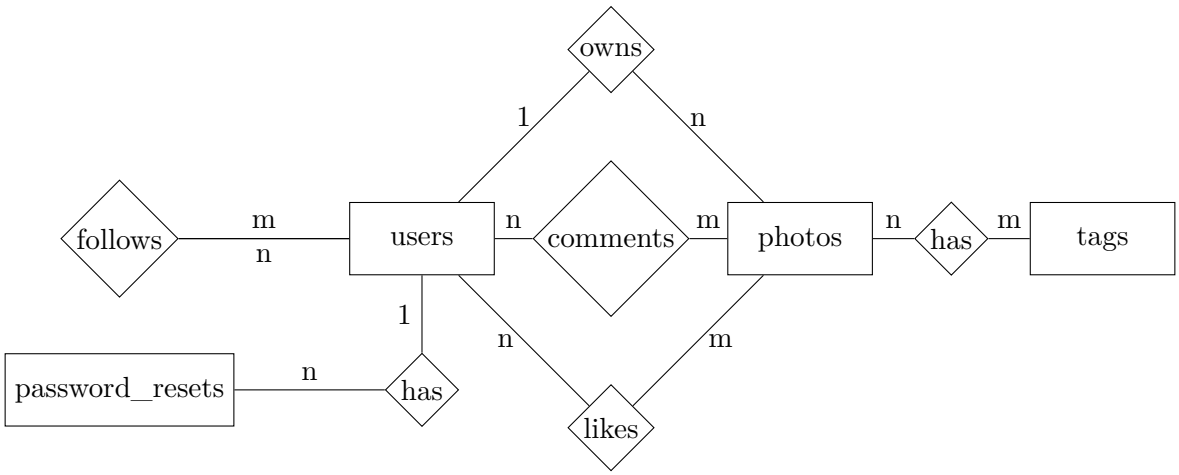


Abbildung 1.7: ERM von InstaHub

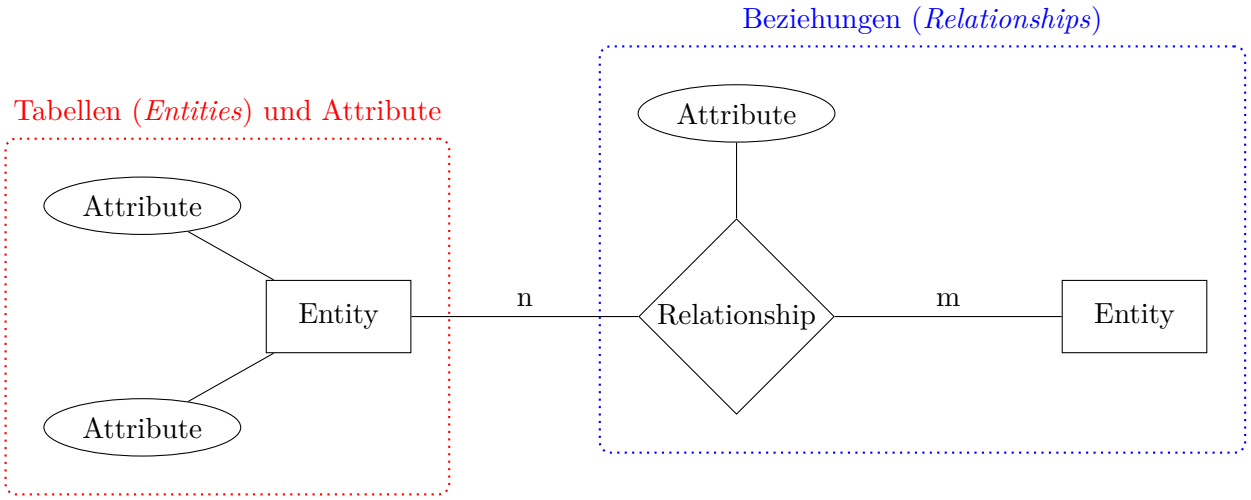


Abbildung 1.8: ERM-Komponenten in der sogenannten Chen-Notation

Die *entities*, die meistens einer Tabelle entsprechen, werden als Rechtecke angegeben. Beziehungen stellen „Verknüpfungen“ zwischen verschiedenen Tabellen dar und sind meist als Raute dargestellt. Die Attribute werden typischerweise als Kreise angegeben. In Abbildung 1.7 sind die Attribute einfachheitshalber nicht angegeben.

Die Zahlen über den Strichen bezeichnen die **Kardinalität** der Beziehungen: Mit der Kardinalität wird ausgedrückt, wie viele Entitäten mit einer anderen Entität in Verbindung stehen können oder

müssen:

- **Eins-zu-Eins (1:1)**: Dies wird normalerweise angezeigt, indem eine „1“ in der Nähe beider Entitäten platziert wird, die durch eine Beziehung verbunden sind. Es bedeutet, dass eine Instanz einer Entität mit genau einer Instanz einer anderen Entität assoziiert ist.
- **Eins-zu-Viele (1:n)**: Dies wird dargestellt, indem eine „1“ in der Nähe der Entität auf der 'eins'-Seite der Beziehung und ein „n“ oder „m“ (für „*many*“=viele) in der Nähe der Entität auf der „viele“-Seite der Beziehung platziert wird. Es zeigt an, dass eine Instanz der ersten Entität mit null, einer oder mehreren Instanzen der zweiten Entität assoziiert sein kann, aber eine Instanz der zweiten Entität nur mit einer Instanz der ersten Entität assoziiert sein kann.
- **Viele-zu-Viele (m:n)**: Dies wird gezeigt, indem ein „m“ oder „n“ in der Nähe beider Entitäten platziert wird. Es zeigt an, dass Instanzen der ersten Entität mit null, einer oder mehreren Instanzen der zweiten Entität assoziiert sein können und umgekehrt.

Folgende Fakten können beispielsweise am ERM in Abbildung 1.7 abgelesen werden:

- Jeder `user` kann beliebig viele (oder 0) `comments` schreiben.
- Jeder `user` kann beliebig vielen (oder 0) anderen Usern followen und kann von beliebig vielen (oder 0) andern usern gefollowt werden.
- Jedes Foto kann beliebig viele (oder 0) Tags haben. Jeder Tag kann auf `n` Fotos angewandt werden.
- Jedes Fotos gehört genau zu einem Benutzerprofil (`owns`). Jedes Benutzerprofil kann beliebig viele (oder 0) Fotos veröffentlichen.
- Jeder `user` kann sein Passwort beliebig viele male (oder 0 mal) zurückgesetzt haben. Jedes des `password_resets` kann aber nur einen `user` betreffen.

Aufgabe 1.36

Eine Schule möchte ihre Datenbankstruktur neu organisieren, um Informationen über Lehrer, Schüler und Klassen besser verwalten zu können. Die Schule hat entschieden, ein Entity-Relationship-Modell (ERM) zu entwerfen, um die Beziehungen zwischen diesen Entitäten klar darzustellen (siehe Abbildung 1.9). Was fällt Ihnen in diesem ERM auf? Stimmen die Kardinalitäten?

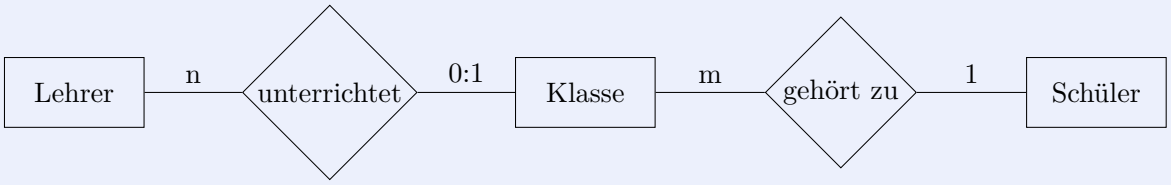


Abbildung 1.9: ERM einer Schul-Datenbank

Lösungsvorschlag zu Aufgabe 1.36

Ein Lehrer kann in diesem ERM nur 0 oder 1 Klasse unterrichten. In Realität ergibt es keinen Sinn, dass ein Lehrer keine Klasse unterrichtet, und auch die Beschränkung auf maximal eine Klasse ergibt keinen Sinn. Statt `0:1` sollte also `m` stehen zwischen „unterrichtet“ und „Klasse“. Zudem sollten `m` und `1` zwischen *Klasse* und *Schüler* getauscht werden.

Aufgabe 1.37

Ein Krankenhaus möchte seine Datenbankstruktur neu organisieren, um Informationen über Ärzte, Patienten und Behandlungen effizienter zu verwalten. Das Krankenhaus hat sich entschieden, ein Entity-Relationship-Modell (ERM) zu entwerfen, um die Beziehungen zwischen diesen Entitäten klar darzustellen (siehe Abbildung 1.10).

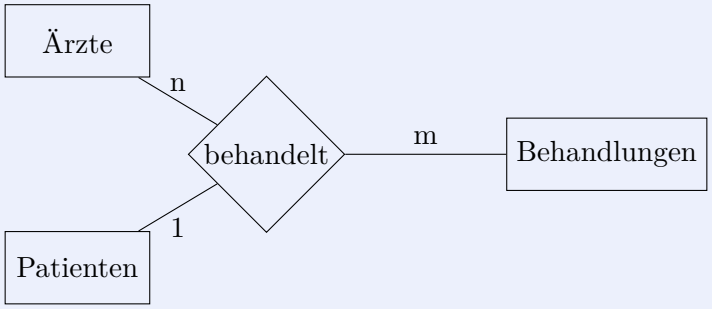


Abbildung 1.10: ERM eines Krankenhaus-Datenbanksystems

Bewerten Sie das dargestellte ERM. Sind die Kardinalitäten korrekt angegeben?

Lösungsvorschlag zu Aufgabe 1.37

In dem dargestellten ERM wird angenommen, dass ein Arzt mehrere Behandlungen durchführen kann (n:m-Beziehung) und dass jeder Patient genau eine Behandlung erhält (1:m-Beziehung). Diese Annahme ist jedoch nicht ganz realistisch, da ein Patient mehrere Behandlungen erhalten könnte. Die Kardinalität zwischen „Patienten“ und „behandelt“ sollte daher m sein, um anzugeben, dass ein Patient mehrere Behandlungen erhalten kann.

1.4 Informationen aus mehrere Tabellen kombinieren

1.4.1 Primärschlüssel

Wie in Unterabschnitt 1.1.3 diskutiert wurde, besitzen Tabellen häufig sogenannte Primärschlüssel (en. *primary keys*). Tabelle 1.6 beispielsweise verwendet einen Primärschlüssel mID (blaue Kolonne, hervorgehoben mit dem Symbol 🔑).

🔑 mID	Name	...
1	Müller	...
2	Schmidt	...
3	Kaufmann	...
...	...	...

Tabelle 1.6: Beispiel-Tabelle: „Angestellte“

Dieser dient in erster Linie dazu, einen Eintrag (in diesem Fall eine Person) eindeutig zu identifizieren. Weshalb ist das nötig?

Zwei Hauptgründe seine angegeben:

- **Eindeutigkeit**: Es könnte sein, dass zwei Personen gleich heissen. In diesem Fall ist es hilfreich, eine eindeutige Identifikationsnummer (oder einen eindeutigen Text) zu haben, um Information zu genau *einer* der beiden Personen abfragen zu können.
- **Dauerhaftigkeit**: Es könnte sein, dass eine Person den Namen wechselt. In diesem Fall möchte man eine eindeutige Identifikationsnummer haben, damit die Informationen zu den Personen wie etwa der Gehalt oder die Abteilung weiterhin eindeutig zur selben Person gehören (und abgefragt werden können).

1.4.2 Fremdschlüssel

Nebst Primärschlüsseln können Tabellen auch sogenannte Fremdschlüssel (en. *foreign keys*) besitzen, die sich auf Primärschlüssel von anderen Tabellen beziehen. Dies erlaubt, Informationen aus mehreren Tabellen zu kombinieren. Gegeben sei folgendes Beispiel mit je einer Tabellen zu Angestellten und Abteilungen (Tabelle 1.7, Tabelle 1.8).

🔑 mID	Name	aID 🔑
1	Müller	31
2	Schmidt	32
3	Kaufmann	32
...	...	...

Tabelle 1.7: Tabelle „**Angestellte**“

🔑 aID	Abteilung
31	Verkauf
32	Technik
33	Marketing
...	...

Tabelle 1.8: Tabelle „**Abteilungen**“

In diesem Beispiel hat die Tabelle „Angestellte“ nicht nur einen Primärschlüssel `mID`, sondern auch einen Fremdschlüssel `aID` (hervorgehoben mit dem Symbol 🔑), der auf den Primärschlüssel mit demselben Name der Tabelle „Abteilungen“ verweist. Angenommen wir möchten nun (mit SQL, nicht mit manuellem Nachschauen) herausfinden, in welcher Abteilung die Person „Kaufmann“ arbeitet, wie lassen sich nun die Informationen aus beiden Tabellen kombinieren?

1.4.3 Tabellen verbinden mit oder ohne `JOIN`

Beispiel 1.16:
Eine erste Möglichkeit, Tabelle 1.7 mit Tabelle 1.8 zu verbinden, besteht darin, einfach beide Tabellen „aufzurufen“:

```
SELECT *
FROM Angestellte, Abteilung
```

mID	Name	Angestellte.aID	Abteilung.aID	Abteilung
1	Müller	31	31	Verkauf
1	Müller	31	32	Technik
1	Müller	31	33	Marketing
1	Müller	31	...	...
2	Schmidt	32	31	Verkauf
2	Schmidt	32	32	Technik
2	Schmidt	32	33	Marketing
2	Schmidt	32	...	...
3	Kaufmann	31	...	...
...	...	...	...	...

Wie wir sehen, wurden Informationen aus beiden Tabellen zusammengeführt, indem alle möglichen Kombinationen beider Tabellen erstellt wurden. Die beiden gleichnamigen Kolonnen erhalten zusätzlich einen Präfix (vorausgehender Text) mit dem Namen der Ursprungstabelle, um diese voneinander abzugrenzen.

Beispiel 1.17:
Wir könnten nun die Auswahl einfach eingrenzen, indem wir zwei Filter einfügen:

```
SELECT *
FROM Angestellte, Abteilung
WHERE Angestellte.aID = Abteilung.aID
AND Name = "Kaufmann"
```

mID	Name	Angestellte.aID	Abteilung.aID	Abteilung
3	Kaufmann	32	32	Technik

Bemerkung 1.1:
Bisher haben wir immer auf Kolonnen von Tabellen zugegriffen, indem wir den Namen der Kolonne geschrieben haben. Wenn wir mit mehreren Tabellen arbeiten, kann es aber passieren, dass Kolonnen denselben Namen haben. Es kann daher nötig sein, zusätzlich zum Kolonnennamen den Namen der Tabelle anzugeben.

Die Tatsache, dass unsere Abfrage mehrere Kolonnen `aID` zurückgibt, mag etwas unschön sein. Dies könnte umgangen werden, indem wir ein `JOIN ... USING`-Konstrukt verwenden.

Beispiel 1.18:
Folgender Befehl führt zu einer einzigen Kolonne `aID` im Endresultat:

```
SELECT *
FROM Angestellte JOIN Abteilung USING (aID)
WHERE Name = "Kaufmann"
```

Achtung

Die Klammer um `aID` muss gesetzt werden, damit der `USING`-Befehl funktioniert.

mID	Name	aID	Abteilung
3	Kaufmann	32	Technik

Bemerkung 1.2:
Statt zwei Tabellen können auch drei oder mehr Tabellen nach den oben erklärten Prinzipien miteinander verbunden werden. Folgendes Beispiel soll Ihnen eine Idee davon geben:

```
SELECT kolonne1, kolonne2, ...
FROM tabelle1
JOIN tabelle2 USING (kolonne_id1)
JOIN tabelle3 USING (kolonne_id2)
...
```

Bemerkung 1.3:
Nach einer Tabellenverbindung können Sie alles machen, was Sie sonst auch tun: filtern mit `WHERE`, gruppieren mit `GROUP BY`, summieren mit `SUM` etc.

Aufgabe 1.38

Lösen Sie Aufgaben 1–10 unter folgendem Link:
https://sql-tutorial.de/home/uebungen.php?lektion=3.

1.5 `JOIN`-Typen

Häufig kann es nützlich sein, mehrere Tabellen auf unterschiedliche Arten miteinander zu verbinden, um neue Einsichten in die Daten zu gewinnen. Verschiedene Tabellen-Verbindungen sind schematisch

in Abbildung 1.11 abgebildet. Die Bedeutung dieser Abbildungen wird in den folgenden Abschnitten erklärt.

(a) `INNER JOIN` (b) `LEFT OUTER JOIN`

(c) `RIGHT OUTER JOIN` (d) `FULL OUTER JOIN`

Abbildung 1.11: Venn-Diagramme unterschiedlicher SQL-Joins

1.5.1 `INNER JOIN` (=`JOIN`)

Die Syntax eines `INNER JOIN`-, oder auch einfach `JOIN`-Befehls sieht wie folgt aus:

```
SELECT kolonne1, kolonne2, ...
FROM tabelle1
INNER JOIN tabelle2
ON tabelle1.kolonne_id1 = tabelle2.kolonne_id2
```

Hier verbinden wir zwei Tabellen, indem wir zwei Kolonnen `kolonne_id1` und `kolonne_id2`, die derselben Information entsprechen (beispielsweise die user-ID), miteinander abgleichen. Beim `INNER JOIN` werden nur die Zeilen aus `tabelle1` retourniert, für die es einen entsprechenden Wert in `kolonne_id2` in `tabelle2` gibt. Alle Werte in `tabelle1` die mit keinem Wert in `tabelle2` überschneiden, sowie umgekehrt, tauchen nicht im Resultat auf (siehe Abbildung 1.12).

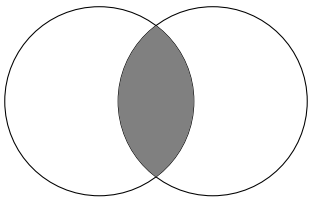


Abbildung 1.12: (`INNER`) `JOIN`

Wichtig zu wissen ist zudem, dass `kolonne1, kolonne2` usw. auf der ersten Zeile nun sowohl von `tabelle1` wie auch `tabelle2` stammen können.

Beispiel 1.19:
Folgender Code gibt die ids aller user zurück, denen der user Mika Kaufmann folgt:

```
SELECT follows.following_id, follows.follower_id, users.name
FROM follows JOIN users
```

```
ON follows.follower_id = users.id
WHERE name="Mika Kaufmann"
```

Wie Sie sehen, kann es hilfreich sein, im `SELECT`-Ausdruck explizit die Tabelle zu nennen, von welcher eine bestimmt Kolonne stammt.

Auch alle anderen SQL-Befehle wie beispielsweise `GROUP BY` können nach einem `JOIN`-Befehl verwendet werden.

Beispiel 1.20:
Folgender Code gibt die Anzahl Fotos für alle Benutzer aus (und sortiert die Resultate absteigend):

```
SELECT name, COUNT(*) AS "Anzahl Fotos"
FROM users
JOIN photos ON users.id = photos.user_id
GROUP BY user_id
ORDER BY `Anzahl Fotos` DESC
```

Achtung

Wie Sie in Beispiel 1.20 sehen, können Kolonnen jederzeit mit dem Befehl `AS "Neuer Name"` umbenannt werden. Allerdings gilt es zu beachten, dass Abstände im Namen eher ungünstig sind: Falls der Name später wiederverwendet wird, wie etwa in einem `ORDER BY`-Ausdruck, müssen sogenannte *backticks* (`) vor und nach dem Namen gesetzt werden.

Aufgabe 1.39

Geben Sie die `id` aller Fotos in der Tabelle `likes` an, die der user Mika Kaufmann gelikt hat.

Lösungsvorschlag zu Aufgabe 1.39

```
SELECT photo_id, user_id, name
FROM likes
INNER JOIN users
ON likes.user_id=users.id
WHERE name="Mika Kaufmann"
```

Aufgabe (Challenge) 1.40

Challenge: Erstellen Sie eine Liste, wo für jedes Hamburger Mitglied die Anzahl seiner Fotos aufgeführt ist. Die Liste soll in absteigender Reihenfolge die Anzahl Fotos pro Mitglieder auflisten.

Lösungsvorschlag zu Aufgabe 1.40

```
SELECT city, username, COUNT(*) AS "Anzahl Fotos"
FROM photos
JOIN users ON photos.user_id = users.id
WHERE city="Hamburg"
GROUP BY username
ORDER BY `Anzahl Fotos` DESC
```

Aufgabe 1.41

Es soll Werbung an alle Strandurlauber verschickt werden. Finden Sie alle Photos die den Hashtag `meer` enthalten. Geben Sie den Namen, die Emailadresse, den Geburtstag und die Stadt der zugehörigen Benutzer aus.

Lösungsvorschlag zu Aufgabe 1.41

```
SELECT description, name, email, birthday, city
FROM photos
JOIN users on photos.user_id = users.id
WHERE description LIKE "%#meer%"
```

Aufgabe 1.42

Geben Sie die 5 User mit den meisten Followern aus.

Lösungsvorschlag zu Aufgabe 1.42

```
SELECT name, COUNT(*) AS "Followers"
FROM users JOIN follows on follows.following_id = users.id
GROUP BY name
ORDER BY Followers DESC
LIMIT 5
```

Aufgabe (Challenge) 1.43

Finden Sie heraus, welche 5 Fotos am meisten Likes erhalten haben. Geben Sie den Benutzernamen und Namen der Ersteller der Fotos an, die id der Fotos sowie die Anzahl Likes.

Lösungsvorschlag zu Aufgabe 1.43

```
SELECT users.username, users.name AS "Name", photos.id AS "Photo id",
    COUNT(likes.photo_id) AS "Likes"
FROM users
JOIN photos ON photos.user_id = users.id
JOIN likes ON photos.id = likes.photo_id
GROUP BY users.name, likes.photo_id
ORDER BY Likes DESC
LIMIT 5
```

1.5.2 `LEFT JOIN` (=`LEFT OUTER JOIN`)

Ein `LEFT JOIN`, oder `LEFT OUTER JOIN` unterscheidet sich von einem `INNER JOIN` dadurch, dass **alle** Einträge der ersten Tabelle (nach dem `SELECT`-Befehl) in der neuen Tabelle erscheinen. Werte aus der zweiten Tabelle, die nicht in der ersten sind, werden jedoch nicht angezeigt (siehe Abbildung 1.13).

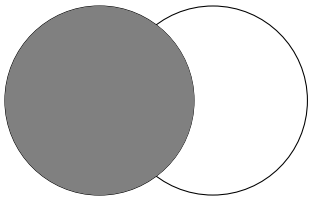


Abbildung 1.13: `LEFT (OUTER) JOIN`

Aufgabe 1.44

Diejenigen BenutzerInnen, die noch keine Fotos hochgeladen habe, sollen per Email dazu aufgefordert werden, Fotos hochzuladen. Finden Sie alle users, die noch keine Fotos hochgeladen haben. Geben Sie deren Name und Email-Adresse aus. Verwenden Sie statt `WHERE` den Befehl `HAVING`.

Lösungsvorschlag zu Aufgabe 1.44

```
SELECT users.name, users.email, COUNT(photos.id) AS "Anzahl Fotos"
FROM users
LEFT JOIN photos ON photos.user_id = users.id
GROUP BY users.name
HAVING `Anzahl Fotos` = 0
```

1.5.3 `RIGHT JOIN` (=`RIGHT OUTER JOIN`)

Ein `RIGHT JOIN` funktioniert analog zu einem `LEFT JOIN` (siehe Abbildung 1.14).

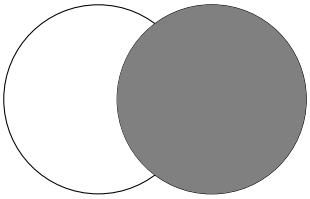


Abbildung 1.14: `RIGHT (OUTER) JOIN`

1.5.4 `FULL JOIN` (=`FULL OUTER JOIN`)

Bei einem `FULL (OUTER) JOIN` werden alle Werte aus der ersten und der zweiten Tabelle angezeigt (siehe Abbildung 1.15).

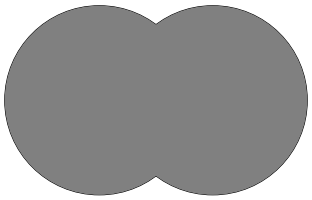


Abbildung 1.15: `FULL (OUTER) JOIN`

In InstaHub funktioniert der `FULL (OUTER) JOIN` nicht, weshalb es an dieser Stelle keine Übungen hierzu gibt.

1.6 Daten bearbeiten mit SQL

Mit SQL kann man Daten nicht nur abfragen, sondern auch erstellen, verändern und löschen. Im folgenden Schauen wir uns dafür nötigen Befehle an.

1.6.1 Eine neue Tabelle erstellen: `CREATE TABLE`

Eine neue Tabelle kann innerhalb einer Datenbank mit dem Befehl `CREATE TABLE` angelegt werden:

```
CREATE TABLE tabellenname(
  kolonne1 eigenschaften1,
  kolonne2 eigenschaften2,
  ...
)
```

Beispiel 1.21:
Die Tabelle `users` wurde mit folgendem Befehl erstellt:

```
CREATE TABLE users (
  `id` int(10) UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
  `username` varchar(191) NOT NULL UNIQUE,
  `email` varchar(191) NOT NULL UNIQUE,
  `password` varchar(191) NOT NULL,
  `name` varchar(191) NOT NULL,
  `bio` varchar(191) DEFAULT NULL,
  `gender` enum('male','female') DEFAULT NULL,
  `birthday` datetime DEFAULT NULL,
  `city` varchar(191) DEFAULT NULL,
  `country` varchar(191) DEFAULT NULL,
  `centimeters` int(11) DEFAULT NULL,
  `avatar` varchar(191) NOT NULL DEFAULT 'avatar.png',
  `role` enum('user','dba','teacher','admin') NOT NULL DEFAULT 'user',
  `is_active` tinyint(1) NOT NULL DEFAULT 0,
  `remember_token` varchar(100) DEFAULT NULL,
  `created_at` timestamp NULL DEFAULT NULL,
  `updated_at` timestamp NULL DEFAULT NULL
)
```

Nach dem Kolonnennamen folgt zuerst der **Kolonnentyp**, d.h., die Angabe, welche Art von Daten in der Kolonne gespeichert werden. Einige der häufigsten Datentypen sind:

- `INT(laenge)` bezeichnet eine Ganzzahl, für die `laenge` bytes zur Verfügung stehen. Beispielsweise speichert die Kolonne `centimeters` eine Zahl von 0 bis 2^{11}, d.h. 2048.
- `TINYINT(1)` speichert den Wert 0 oder 1, wobei 0 typischerweise „falsch“ und 1 „wahr“ bedeuten.
- `DATETIME` ist ein Datum mit einer Uhrzeit
- `VARCHAR(laenge)` bezeichnet eine Zeichenkette, d.h. ein Text, der mit maximal `laenge` vielen Bit kodiert wird.
- `TEXT` bezeichnet eine Zeichenkette mit maximaler Länge von 65’535 bytes.
- `ENUM('wert1', 'wert2',...)` bezeichnet eine Auflistung (engl. *enumeration*) möglicher Werte. Beispielsweise können nur „male“ und „female“ im Feld `gender` eingetragen werden.

Zudem können folgende Eingeschaften für Kolonnen angegeben werden:

- `NOT NULL`: Feld darf nicht leer sein. Falls kein `DEFAULT`-Wert angegeben wird, muss, der Wert für diese Kolonne beim Erstellen eines neuen Benutzerkontos immer mitgeliefert werden.
- `DEFAULT wert`: Standard-Wert, falls nichts anderes angegeben wird. Das Attribut `avatar` nimmt beispielsweise den Wert `avatar.png` an, falls kein anderes Bild hochgeladen wird.
- `AUTO_INCREMENT`: Zahl, die bei jedem neuen Eintrag automatisch immer um 1 grösser wird
- `PRIMARY KEY`: Primärschlüssel (siehe Tabelle 1.3).
- `FOREIGN KEY (kolonneID) REFERENCES tabelle2(kolonneID2)`: Fremdschlüssel (siehe Tabelle 1.3).
- `UNIQUE`: Darf keine doppelten Werte enthalten
- `UNSIGNED`: Ohne Vorzeichen (+-), also nur positive Zahlen

Achtung

Je nach SQL-Version muss ein Fremdschlüssel erst *nach dem Erstellen einer Tabelle* als Fremdschlüssel deklariert werden. Dies kann mit dem Befehl `ALTER TABLE` gemacht werden.

Beispiel 1.22:
In folgendem Beispiel wird in Tabelle `table2` das Attribut `fID` nachträglich als Fremdschlüssel deklariert, indem es auf das Attribut `ID` bei `table1` zeigt.

```
CREATE TABLE table1 (
    ID int(10) UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
    ...
);

CREATE TABLE table2 (
    `ID` int(10) UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
    `fID` int(10) UNSIGNED NOT NULL
    ...
);

ALTER TABLE table2
ADD FOREIGN KEY (fID) REFERENCES table1(ID)
```

Achtung

Auf InstaHub kann immer nur ein Befehl pro Mal ausgeführt werden, das „;“-Zeichen funktioniert also leider nicht. Stattdessen müssen Sie in InstaHub jeden Befehl einzeln eingeben.

Aufgabe 1.45

Schreiben Sie die SQL-Befehle, um Tabelle 1.7 und Tabelle 1.8 zu erstellen. Sie müssen dabei lediglich eine leere Tabelle mit den korrekten Kolonnen-Namen und Kolonnen-Typen (Zahl, Text, etc.) erstellen, Daten müssen Sie noch keine einfügen.

Lösungsvorschlag zu Aufgabe 1.45

```
CREATE TABLE abteilungen (
    aID int(10) UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
    abteilungsname varchar(191) NOT NULL UNIQUE
)
```

```
CREATE TABLE angestellte (
    `mID` int(10) UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
    `name` varchar(191) NOT NULL UNIQUE,
    `aID` int(10) UNSIGNED NOT NULL
)
```

```
ALTER TABLE angestellte
ADD FOREIGN KEY (aID) REFERENCES abteilungen(aID)
```

1.6.2 Tabellen verändern: `ALTER`

Mit dem Befehl `ALTER tabellenname` können Tabellen und deren Attribute nachträglich, d.h. nach der Erstellung der Tabelle, verändert werden.

Beispiel 1.23:
Um eine weitere Kolonne hinzuzufügen, kann folgender Befehl verwendet werden:

```
ALTER TABLE tabellenname
ADD neue_kolonne eigenschaften
```

Beispiel 1.24:
Um eine Kolonne umzubenennen, kann folgender Befehl verwendet werden:

```
ALTER TABLE tabellenname
RENAME COLUMN alter_name TO neuer_name;
```

Achtung

Beim Umbenennen einer Kolonne können Datenbankschemen (z.B. Fremschlüssel-Referenzen) zerstört werden, z.B. falls ein Primärschlüssel umbenannt wird, welcher von einer anderen Tabelle per Fremschlüssel referenziert wird (siehe Unterabschnitt 1.6.1).

1.6.3 Tabellen löschen: `DROP TABLE`

Mit dem Befehl `DROP TABLE tabellenname` kann eine Tabelle komplett löschen.

Achtung

Dieser Schritt ist irreversibel. Falls Sie eine Tabelle irrtümlich löschen, geben Sie mir Bescheid, damit ich Ihre Datenbank zurücksetzen kann.

Aufgabe 1.46

Erstellen Sie auf InstaHub eine Tabelle `users2` mit denselben Eigenschaften wie `users` (siehe Beispiel 1.21)

Lösungsvorschlag zu Aufgabe 1.46

```
CREATE TABLE users2 (
  `id` int(10) UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
  `username` varchar(191) NOT NULL UNIQUE,
  `email` varchar(191) NOT NULL UNIQUE,
  `password` varchar(191) NOT NULL,
  `name` varchar(191) NOT NULL,
  `bio` varchar(191) DEFAULT NULL,
  `gender` enum('male','female') DEFAULT NULL,
  `birthday` datetime DEFAULT NULL,
  `city` varchar(191) DEFAULT NULL,
  `country` varchar(191) DEFAULT NULL,
  `centimeters` int(11) DEFAULT NULL,
  `avatar` varchar(191) NOT NULL DEFAULT 'avatar.png',
  `role` enum('user','dba','teacher','admin') NOT NULL DEFAULT 'user',
  `is_active` tinyint(1) NOT NULL DEFAULT 0,
  `remember_token` varchar(100) DEFAULT NULL,
  `created_at` timestamp NULL DEFAULT NULL,
  `updated_at` timestamp NULL DEFAULT NULL
)
```

Aufgabe 1.47

Löschen Sie danach die Tabelle `users2` wieder.

Lösungsvorschlag zu Aufgabe 1.47

```
DROP TABLE users2
```

1.6.4 Daten einfügen: `INSERT INTO`

Mit `INSERT INTO` können neue Zeilen in eine bestehende Tabelle eingefügt werden.

Die Syntax von `INSERT INTO` sieht wie folgt aus:

```
INSERT INTO tabelle_name (kolonne1, kolonne2, kolonne3, ...)
VALUES (wert1, wert2, wert3, ...)
```

Beispiel 1.25:
Folgender Code fügt eine neue Person in die Tabelle `users` ein:

```
INSERT INTO users (username, email, password, name, bio, gender, birthday,
    city, country, centimeters, avatar, role, is_active, remember_token,
    created_at, updated_at)
VALUES ('guenther37', 'guenther@instahub.test', '12345', 'Günther Müller', '
    Günther mag Kartoffelsalat.', 'male', '2006-06-06 00:00:00', 'Leipzig', '
    Deutschland', '173', 'avatar.png', 'user', '0', NULL, now(), now())
```

Wie Sie sehen, wird zuerst die Tabelle (`users`) aufgelistet, danach die Kolonnen und schliesslich die Werte.

Aufgabe 1.48

Für welche Kolonnen müssen für einen neuen Benutzer immer die Werte mitgegeben werden? S. die Erklärungen zu `NOT NULL` und `DEFAULT` in Unterabschnitt 1.6.1.

Lösungsvorschlag zu Aufgabe 1.48

`username, email, password, name, role, is_active`

Aufgabe 1.49

Erstellen Sie einen neuen Eintrag in der Tabelle `users` mit folgenden Eigenschaften:

Kolonne	Wert
id	300
username	testuser
email	test@test.com
password	test123
name	Test-Vorname Test-Nachname
role	user
is_active	1

Überprüfen Sie danach, ob der Eintrag korrekt erstellt worden ist, indem Sie die Daten des users `testuser` abfragen (alle Attribute).

Lösungsvorschlag zu Aufgabe 1.49

```
INSERT INTO users (id, username, email, password,
                   name, role, is_active)
VALUES (300, 'testuser', 'test@test.com', 'test123', 'Test-Vorname Test
    -Nachname', 'user', '1')
```

Das Resultat kann mit einer zweiten Abfrage überprüft werden:

```
SELECT * FROM users WHERE username="testuser"
```

1.6.5 Daten verändern: `UPDATE`

Mit `UPDATE` können Werte einer Tabelle aktualisiert werden. Die Syntax von `UPDATE` sieht wie folgt aus:

```
UPDATE tabelle_name
SET kolonne1 = wert1, kolonne2=wert2, ...
WHERE bedingung(en)
```

Aufgabe 1.50

Ersetzen Sie die Stadt „Berlin“ überall durch „Bern“

Lösungsvorschlag zu Aufgabe 1.50

```
UPDATE users
SET city="Bern"
WHERE city="Berlin"
```

Aufgabe 1.51

Setzen Sie die Körpergrössen aller männlichen Mitglieder auf 190. Überprüfen Sie danach die Richtigkeit Ihres Befehls, die Kolonnen für die Körpergrösse, das Geschlecht und den Name aller männlichen Mitglieder abfragen.

Lösungsvorschlag zu Aufgabe 1.51

```
UPDATE users
SET centimeters=190
WHERE gender="male"
```

Um zu überprüfen, ob der Befehl funktioniert hat, können Sie danach einfach die Tabelle nochmals anzeigen:

```
SELECT centimeters, gender, name
FROM users
WHERE gender="male"
```

1.6.6 Einträge löschen: `DELETE FROM`

Mit dem Befehl `DELETE FROM` können Einträge aus einer Tabelle gelöscht werden.

```
DELETE FROM tabellenname
WHERE bedingungen
```

Beispiel 1.26:
Folgender Code löscht den Eintrag für user `guenther37`:

```
DELETE FROM users
WHERE username="guenther37"
```

Aufgabe 1.52

Löschen Sie den Eintrag, den Sie in Aufgabe 1.49 erstellt haben. Überprüfen Sie, ob der Eintrag verschwunden ist, indem Sie danach `SELECT * FROM users` ausführen.

Lösungsvorschlag zu Aufgabe 1.52

```
DELETE FROM users
WHERE username="testuser"
```

1.7 Weiterführende Links und Übungen

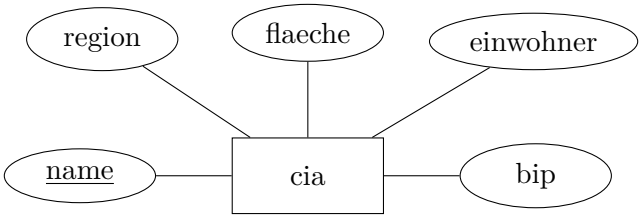
Folgende Links dienen der Vertiefung in das Thema SQL (und der Prüfungsvorbereitung).

Allgemein zu Datenbanken:

- https://oinf.ch/kurs/vernetzung-und-systeme/datenbanken/
- https://wi-wissen.github.io/instahub-doc-de/#/exercices

Zu SQL:

- https://www.w3schools.com/sql/default.asp (auf Englisch)
- https://sql-island.informatik.uni-kl.de
- https://sql-tutorial.de/home/lektionen.php?lektion=1. Hier wird unter anderem folgende Datenbank verwendet:



Lernziele: SQL

- ☐ Ich kann zwischen Bits und Bytes unterscheiden und kenne die Bedeutungen von gängigen Grösseneinheiten (Kilo-, Mega-, Giga-, Tera- und Petabyte).
- ☐ Ich weiss betreffend eine Tabelle, was eine Attribut und was eine Zeile ist.
- ☐ Ich kann einfache SQL-Abfragen formulieren mit Befehlen wie `SELECT`, `WHERE`, `ORDER BY`, `LIMIT` und weiteren Befehlen (siehe Cheatsheet für vollständige Liste aller zu lernenden SQL-Befehle).
- ☐ Ich kann Aggregatsfunktionen wie `SUM`, `MIN`, `MAX`, `COUNT` oder `LENGTH` auf einzelne Kolonnen anwenden und kenne deren Bedeutung.
- ☐ Ich kann die oben erwähnten SQL-Begriffe gruppenweise anwenden, indem ich sie mit dem Befehl `GROUP BY` kombiniere.
- ☐ Ich kann den Unterschied zwischen einem Primär- und einem Fremdschlüssel benennen und erklären.
- ☐ Ich kann anhand einer Übersicht von Tabellen in einer Datenbank erkennen, welche Attribute Primär- und welche Sekundärschlüssel sind.
- ☐ Ich kann Informationen aus mehreren Tabellen kombinieren, indem ich den `JOIN`-Befehl verwenden.
- ☐ Ich kann Tabellenverbindungen sowohl mit dem Befehl `ON` wie `USING` erstellen und verstehe, welcher Begriff in welchen Situationen besser geeignet ist.
- ☐ Ich kann aus kombinierten Tabellen neue Informationen gewinnen, indem ich die bereits erwähnten SQL-Codewörter wie z.B. `GROUP BY` in Kombination mit einem einem `JOIN` verwende.
- ☐ Ich kann Tabellen erstellen, löschen und verändern, indem ich (unter anderem) die Befehle `CREATE TABLE`, `DROP TABLE` und `ALTER TABLE` verwenden.
- ☐ Ich kann Einträge (Zeilen) in einer existierenden Tabelle hinzufügen, verändern oder löschen, indem ich die Befehle `INSERT INTO`, `UPDATE` und `DELETE FROM` verwende.

Cheatsheet

Folgendes Cheatsheet ist basiert auf der Datenbank der Social-Media-Plattform InstaHub. In den ersten Befehlen wird insbesondere die Tabelle `users` verwendet. Die Tabelle enthält unter anderem folgende Informationen:

id	username	name	birthday	city	centimeters
1	niclas258	Niclas Schweizer	2001-01-31	Wremen	182
2	rafael54	Rafael Probst	2004-08-06	Leipzig	187
3	luis52	Luis Krüger	2004-12-15	Lautertal	173
...	...	...	...	...	...

Daneben enthält die Tabelle noch viele weitere Spalten mit Informationen zum Land, Geschlecht usw.

Spalten auswählen: SELECT

Auswahl aller Spalten der Tabelle:

```
SELECT *
FROM users
```

Auswahl der Spalten `city` und `gender` der Tabelle `users`:

```
SELECT city, gender
FROM users
```

Duplikate löschen: DISTINCT

Mit dem Zusatz `DISTINCT` werden Duplikate aus den Resultaten gelöscht.

```
SELECT DISTINCT gender
FROM users
```

Sortieren: ORDER BY

Mit dem Befehl `ORDER BY` können Resultate sortiert werden, nach einer oder mehreren Spalten. Mit folgendem Befehl erhalten wir die Namen aller Users, sortiert nach Stadt und nach Region:

```
SELECT name, city
FROM users
ORDER BY city ASC, name ASC
```

- `DESC` = absteigend (*descending*)
- `ASC` = aufsteigend (*ascending*)

Erste / Letzte Zeilen: LIMIT

Mit dem Befehl `LIMIT` können eine Tabelle auf die ersten n Zeilen beschränkt werden. Z.B. können die Namen der ersten drei Personen der Tabelle `users` wie folgt abgefragt werden:

```
SELECT name
FROM users
LIMIT 3
```

Zeilen Filtern: WHERE

Mit dem Befehl `WHERE` können die Resultate einer Abfrage nach eigenen Kriterien gefiltert werden:

```
SELECT name, city, gender
FROM users
WHERE gender='male'
```

Operatoren	Bedeutung
=	gleich (=)
<>	ungleich ($\neq$)
<	kleiner als (<)
<=	kleiner oder gleich ($\leq$)
>	grösser als (>)
>=	grösser oder gleich ($\geq$)
`BETWEEN x AND y`	ein Wert zwischen x und y
`IN (wert1, wert2, ...)`	einer von mehreren Werten
`IS NULL`	Wert ist leer

Filter kombinieren: AND, OR

Mehrere Filter können mit `AND` („und“ → beides muss wahr sein) oder `OR` („oder“ → eine der beiden Bedingungen muss wahr sein) verbunden werden.

Users aus Leipzig, Berlin oder Hamburg, welche grösser als 170 sind:

```
SELECT name, city
FROM users
WHERE city IN ('Leipzig','Berlin',
    'Hamburg')
 AND centimeters > 170
```

Ungefähre Treffer: LIKE

Mit folgendem Befehl erhalten wir alle Städte, die mit „Be...“ beginnen:

```
SELECT DISTINCT city
FROM users
WHERE city LIKE 'Be%'
```

Das Prozentzeichen (%) ist ein *Platzhalter* und steht für „irgendetwas kommt hier (oder nicht)“. In diesem Beispiel bedeutet das, dass 0, 1, oder mehr Zeichen auf das „Be...“ folgen können.

Aggregatsfunktionen

Mit Aggregatsfunktionen werden Daten zusammengefasst, beispielsweise um die Anzahl Zeilen (`COUNT(*)`), die Anzahl nicht leerer Zellen in einer Spalte (`COUNT(spalte)`) oder die Anzahl unterschiedlicher Werte in einer Spalte (`COUNT(DISTINCT spalte)`) zu zählen oder um das Maximum, das Minimum, die Summe oder den Mittelwert einer Spalte zu berechnen (`MAX(spalte)` / `MIN(spalte)` / `SUM(spalte)` / `AVG(spalte)`). Z.B. berechnet folgender Ausdruck die Anzahl Users in Leipzig:

```
SELECT city, COUNT(*)
FROM users
WHERE city = 'Leipzig'
```

Rechnen und umbenennen: AS

Mit Spalten sowie Aggregatsfunktionen kann man rechnen, beispielsweise um ein Resultat durch eine andere Zahl zu dividieren. Zudem können Spalten-Titel mit dem Befehl `AS` umbenannt werden:

```
SELECT centimeters/100 AS 'Grösse
    in Metern'
FROM users
```

Gruppieren: GROUP BY

Aggregatsfunktionen können auch pro Gruppe verwendet werden, z.B. um die Anzahl Mitglieder in jeder Stadt zu berechnen:

```
SELECT city, COUNT(*) AS 'Users
    pro Stadt'
FROM users
GROUP BY city
```

Filtern *nach* Gruppieren: HAVING

Mit `HAVING` können Resultate *nach dem* Gruppieren gefiltert werden. `WHERE` filtert *vor* dem Gruppieren und steht demnach immer vor einem `GROUP BY`.

Durchschnittliche Körpergrösse aller männlichen Mitglieder in jeder Stadt berechnen, danach auf Städte beschränken, in denen die Menschen durchschnittlich zwischen 150 und 155 gross sind:

```
SELECT city, AVG(centimeters) AS '
    Körpergrösse'
FROM users
WHERE gender = 'male'
GROUP BY city
HAVING `Körpergrösse` BETWEEN 150
    AND 155
```

Texte werden immer innerhalb von Anführungszeichen (`''`) geschrieben. Spaltennamen, welche Spezialzeichen oder Abstände enthalten, werden innerhalb von *backticks* (``` `` ```) geschrieben.

Tabellen erstellen: CREATE TABLE

Eine Tabelle `meinetabelle` kann wie folgt erstellt werden:

```
CREATE TABLE meinetabelle(
  name_spalte1 spaltentyp1,
  name_spalte2 spaltentyp2,
  ... --etc.
)
```

Gängige Spalten-Typen sind:

`INTEGER`	Ganzzahl Z.B. 35
`REAL`	Kommazahl Z.B. 3.341
`DATE`	Datum Format: `'YYYY-MM-DD'`
`BOOLEAN`	Wahrheitswert Wahr (1) oder Falsch (0)
`VARCHAR(n)`	Text n = maximale Länge

Tabellen ändern: ALTER

Mit dem Befehl `ALTER tabellenname` können Tabellen verändert werden, beispielsweise um eine Spalte hinzuzufügen oder umzubenennen.

```
ALTER TABLE tabellenname
ADD neue_spalte eigenschaften
```

```
ALTER TABLE tabellenname
RENAME COLUMN name_vorher TO
    name_neu;
```

Tabellen löschen: DROP TABLE

```
DROP TABLE tabellenname
```

Überprüfen, ob existiert: IF EXISTS

Wenn man eine Tabelle erstellen will, welche es bereits gibt, kann eine Fehlermeldung erscheinen. Folgender Befehl wird nur ausgeführt, falls es noch keine Tabelle `testtabelle` gibt:

```
CREATE TABLE IF NOT EXISTS
    testtabelle(
  name_spalte1 spaltentyp1,
  name_spalte2 spaltentyp2,
  ... --etc.
)
```

Ähnlich kann vor man den Befehl `DROP TABLE` anpassen, so dass er nur ausgeführt wird, wenn es eine solche Tabelle wirklich gibt:

```
DROP TABLE IF EXISTS testtabelle
```

Daten einfügen: INSERT INTO

Neue Einträge (Zeilen) können mit dem Befehl `INSERT INTO` in eine Tabelle eingefügt werden. Beispielsweise können mit folgender Befehlsstruktur drei neue Zeilen eingefügt werden:

```
INSERT INTO tabellenname
    (spalte1, spalte2, ...)
VALUES
    (wert1, wert2, ...),
    (wert1, wert2, ...),
    (wert1, wert2, ...)
```

Zuerst werden also die Spalten angegeben, für welche Werte eingefügt werden, danach die Werte für jede neue Zeile. Werte in nicht angegebenen Spalten bleiben leer.

Daten verändern: UPDATE

Existierende Daten können wie folgt geändert werden:

```
UPDATE tabelle_name
SET spalte1 = wert1, ...
WHERE bedingung(en)
```

Beispielsweise können mit folgendem Befehl alle Users, die zur Stadt Berlin gehören, der Stadt Bern zugeordnet werden:

```
UPDATE users
SET city='Bern'
WHERE city='Berlin'
```

Einträge löschen: DELETE FROM

Einzelne Einträge (Zeilen) können mit einer `WHERE`-Bedingung und dem Befehl `DELETE FROM` gelöscht werden:

```
DELETE FROM users
WHERE username='guenther37'
```

Mehrere Tabellen verbinden: JOIN

Mehrere Tabellen können mit folgendem Befehl zu einer Tabelle verbunden werden:

```
SELECT t1.spalte_x, t2.spalte_y,
    ...
FROM tabelle1 AS t1
JOIN tabelle2 AS t2
ON t1.spalte_id1 = t2.spalte_id2
```

Falls Primär- und Fremdschlüssel in beiden Tabellen gleich heissen, kann man `USING` verwenden:

```
SELECT *
FROM tabelle1
JOIN tabelle2 USING (spalte_id)
```

Unterabfragen (Subqueries)

Unterabfragen (en. *subqueries*) können wie folgt geschrieben werden:

```
SELECT spalte11
FROM tabelle1
WHERE spaltenname IN
    (SELECT spalte2 FROM tabelle2
    WHERE ...);
```

Dabei können alle bekannten Vergleichsoperatoren wie `=`, `IN`, `>`, `<`, usw. verwendet werden.

Mehrere Ausdrücke verbinden

Mehrere SQL-Ausdrücke können mit Semikolon (`;`) verbunden werden:

```
-- Tabelle erstellen, danach
    wieder löschen
CREATE TABLE IF NOT EXISTS
    testtabelle(
  name_spalte1 spaltentyp1,
  name_spalte2 spaltentyp2,
  ... --etc.
);
DROP TABLE IF EXISTS testtabelle
```

Kommentare

Kommentare werden von SQL ignoriert und dienen der besseren Leserlichkeit des Codes. Kommentare werden durch zwei Bindestriche gekennzeichnet (siehe Box zu „Mehrere Ausdrücke verbinden“)

Weitere Befehle & Feedback

Weitere SQL-Befehle und Erklärungen finden Sie unter w3schools.com.

Verbesserungsvorschläge können gerne an Cyril Wendl geschickt werden.

Glossar

ERM Entity-Relationship Model. 22, 23, 27, 28

SQL Structured Query Language. 6, 8, 10–12, 14, 16, 20, 24, 38, 44, 47