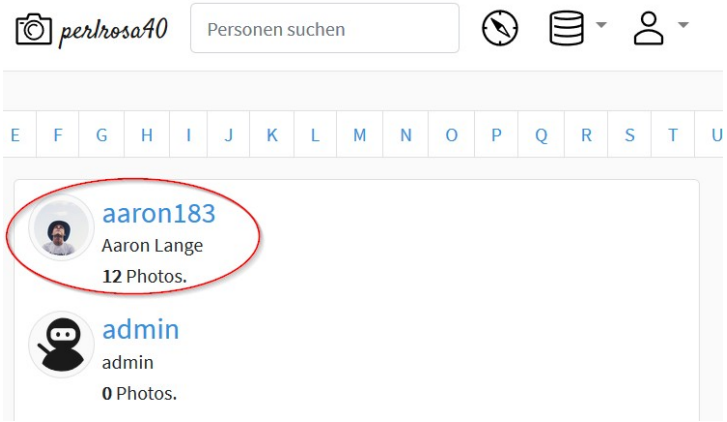
1 SQL-Abfragen über mehrere Tabellen (JOIN)

Zuletzt hatten Sie in Kapitel 9 mit SQL gearbeitet. Sie hatten die neue Tabelle `photos` erstellt und mit Inhalt gefüllt. Wir kommen nun zurück zu SQL und zu `SELECT` und starten erst einmal in einer Aufwärmübung:

`aaron183` ist einer Ihrer Benutzer. In der Benutzeransicht sehen Sie, dass er 12 Fotos eingestellt hat:



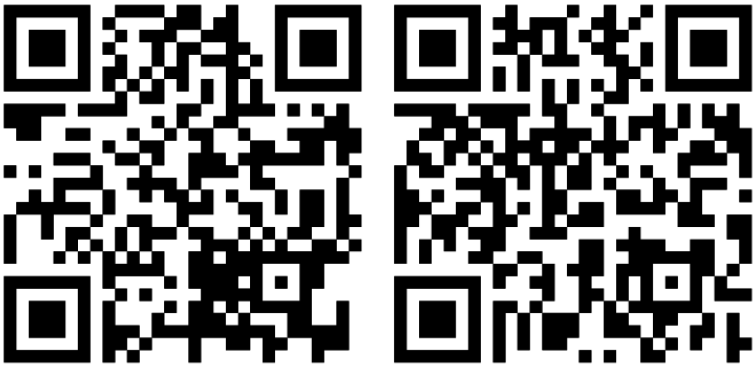
Aaron hat 12 Fotos eingestellt

Wie bekommt InstaHub eigentlich heraus, dass `aaron183` 12 Fotos hochgeladen hat? In der Tabelle `photos` enthält das Feld `user_id` Werte, die in der Tabelle `users` in der Spalte `id` stehen. In Zukunft schreiben wir statt "*Die Spalte* `user_id` *in der Tabelle* `photos`" einfach kurz "`photos.user_id`."

Aufgabe 11.1: Aarons Fotos

1. *Erklären* Sie das Konzept der Primär- und Fremdschlüssel an Hand des obigen Beispiels!
2. *Erstellen* Sie einen `SELECT`-Befehl der in der Tabelle `users` die `id` des Users mit dem Benutzernamen `aaron183` ermittelt!
3. *Erstellen* Sie einem zweiten `SELECT`-Befehl, der in der Tabelle `photos` die Zahl der von `aaron183` hochgeladenen Fotos ermittelt!

Die folgenden QR-Codes enthalten Lösungsvorschläge für diese Aufgabe:



Lösungsvorschläge für Aufgabe 11.1

1.1 Zwei Tabellen verbinden

Nach diesem Wiedereinstieg in SELECT stellen wir uns natürlich die Frage, ob man immer zwei SELECT-Befehle braucht.

Braucht man nicht. Man kann in der FROM-Klausel auch eine *Liste von Tabellen* angeben:

Aufgabe 11.2 Erster Versuch, zwei Tabellen zu verbinden

Geben Sie den folgenden SQL-Befehl ein:

```
SELECT  COUNT(username)
FROM    users, photos   -- 2 Tabellen in der FROM-Klausel!
WHERE   username="aaron183"
```

1. *Beschreiben* Sie das Ergebnis.
2. *Begründen* Sie welches Ergebnis Sie ungefähr erwartet hätten.
3. *Probieren* Sie einmal folgendes: *Ermitteln* Sie mittels `COUNT` die Zahl der Datensätze der Tabellen `users` und `photos`. *Ermitteln* Sie dann mittels `COUNT` die Zahl der Datensätze, die herauskommen, wenn Sie im obigen `SELECT` die `WHERE`-Klausel weglassen. In welcher mathematischen Beziehung stehen diese drei Zahlen?

1.2 (INNER) JOIN

Wir müssen SQL sagen, dass nur die zu einander passenden Datensätze aus den beiden Tabellen miteinander kombiniert werden sollen. Das geht so:

```
SELECT  COUNT(username)
FROM    users, photos
WHERE   users.id = photos.user_id   -- Nur die passenden!
        AND username = "aaron183"   -- Nur die von Aaron!
```

In beiden Tabellen gibt es eine Spalte `id`. Woher soll SQL wissen, welche

Spalte wir meinen? Daher muss bei `users.id` die Tabelle angegeben werden. Die Spalte `user_id` gibt es (derzeit) nur einmal daher muss hier ebenso wenig wie bei `username` eigentlich keine Tabelle angegeben werden. In der Abfrage oben wurde die Tabelle `photos` aus optischen Gründen angegeben.

Man kommt recht weit mit dieser Art, Abfragen über mehrere Tabellen zu verfassen. Aber es geht übersichtlicher!

Aufgabe 11.3: JOIN analysieren

Betrachten Sie das folgende SQL-`SELECT` mit dem neuen Schlüsselwort `JOIN` (engl. *to join:* sich verbinden):

```
SELECT  count(users.id)
FROM    users JOIN photos
        ON (users.id = user_id)
WHERE   username="aaron183"
```

1. *Markieren* Sie in dem obigen `SELECT`-Befehl alle Stellen, die neue Sprachelemente enthalten!
2. *Begründen* Sie warum die Lösung mit `JOIN` übersichtlicher ist als die Lösung ohne `JOIN`. (Wenn Sie noch nicht restlos überzeugt sind: Sie werden gleich noch sehen, dass Sie mit `JOIN` mächtiger als ohne `JOIN` sind.)
3. *Probieren* Sie den Befehl aus!

Aufgabe 11.4: JOIN anwenden

Entwickeln Sie jeweils eine SQL-Abfrage mit `JOIN`!

1. Welche Fotos haben die Mitglieder aus Hamburg hochgeladen?
2. Welches sind die Top-10 der Mitglieder, bezogen auf die Zahl der Fotouploads?
3. Zu jedem Mitglied ist die Zahl der von ihm*ihr hochgeladenen Fotos gesucht.

1.3 Mehr Tabellen für InstaHub! (Tabellenmodelle)

Um komplexere Abfragen behandeln zu können, benötigen wir ein wenig mehr Tabellen.

Bisher fehlen InstaHub noch folgende für soziale Netze wichtige Funktionen:

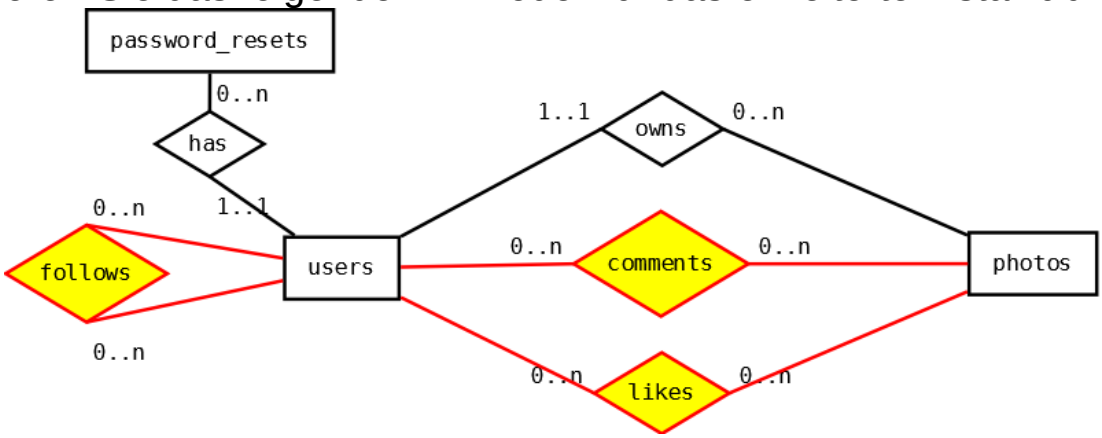
1. Man will anderen Nutzer*innen *folgen* können.

2. Man will Fotos anderer Nutzer*innen* liken* können.
3. Man will Fotos anderer Nutzer*innen* kommentieren* können.

Diese Funktionen bekommt InstaHub jetzt!

Aufgabe 11.5: ER-Modell von InstaHub (2)

Analysieren Sie das folgende ER-Modell für das erweiterte InstaHub!



InstaHub-ERM Version 2

1. *Beurteilen* Sie die vorgeschlagenen Kardinalitäten!
2. *Beurteilen* Sie ob das neue Modell die versprochenen neuen Funktionen ermöglicht!

Jeder Datensatz in der Tabelle `users` kann mit jedem Datensatz in der Tabelle `photos` verbunden werden. Und umgekehrt kann jeder Datensatz in der Tabelle `photos` kann mit jedem Datensatz in der Tabelle `users` verbunden werden. Eine solche Beziehung nennt man auch n:m-Beziehung.

n:m-Beziehungen werden in der Datenbank mit Hilfe einer neuen Tabelle realisiert, die die Primärschlüssel der verbundenen Tabellen als Fremdschlüssel enthält, sie enthält also *zwei Fremdschlüssel*. Die Relation `likes` könnte als Tabelle so aussehen:

user_id	**photo_id**
15	12
15	14
18	14

In diesem Beispiel hat die*der Benutzer*in mit der `user_id` 15 die Fotos 12 und 14 geliked. Das Foto 14 gefällt auch der*dem Benutzer*in mit der id 18.

Einfacher notiert man Tabellen in der folgenden Form, dem so genannten *Tabellenmodell*:

likes(^ , ^)

Der Pfeil bedeutet, dass es sich um einen Fremdschlüssel handelt, unterstrichene Attribute sind Teil des Primärschlüssels. Tabellenmodelle sind

nicht genormt, daher werden Sie viele verschiedene Notationen finden.

Anders als ER-Modelle enthalten Tabellenmodelle auch die Fremdschlüssel. Es ist vielfach hilfreich, bei der Umsetzung eines ER-Modells in eine Datenbank das ER-Modell zunächst in ein Tabellenmodel zu überführen.

Aufgabe 11.6: Die Tabellen konstruieren (1)

Entwickeln Sie für die neuen Relationships `comments` und `follows` ein Tabellenmodell!

Aufgabe 11.7: Die Tabellen konstruieren (2)

Entwickeln Sie für jede der drei neuen Tabellen einen SQL-CREATE-Befehl!

Bevor Sie Ihren Befehl abschicken, vergleichen Sie Ihre Lösung unbedingt mit dem Lösungsvorschlag Ihrer Lehrkraft!

Tipp: Gehen Sie dabei wie folgt vor:

1. *Bestimmen* Sie die nötigen Attribute.
2. *Bestimmen* Sie zu jedem Attribut den passenden Datentyp sowie einen Default-Wert und legen Sie fest, ob `NULL` erlaubt sein soll.
3. *Geben* Sie die *Primärschlüssel* an.
4. *Fügen* Sie die *Fremdschlüssel* hinzu.

Aufgabe 11.8: Die Tabellen konstruieren (3)

Setzen Sie Ihre `Create`-Befehle aus der vorherigen Aufgabe ab! *Überlegen* Sie zuvor, ob die Reihenfolge Ihrer Befehle eine Rolle spielt!

Aufgabe 11.9: Die Tabellen füllen

Füllen Sie die neuen Tabellen mit den Daten, die Ihre Lehrkraft für Sie bereit gestellt hat. *Überlegen* Sie zuvor, ob die Reihenfolge Ihrer Befehle eine Rolle spielt!

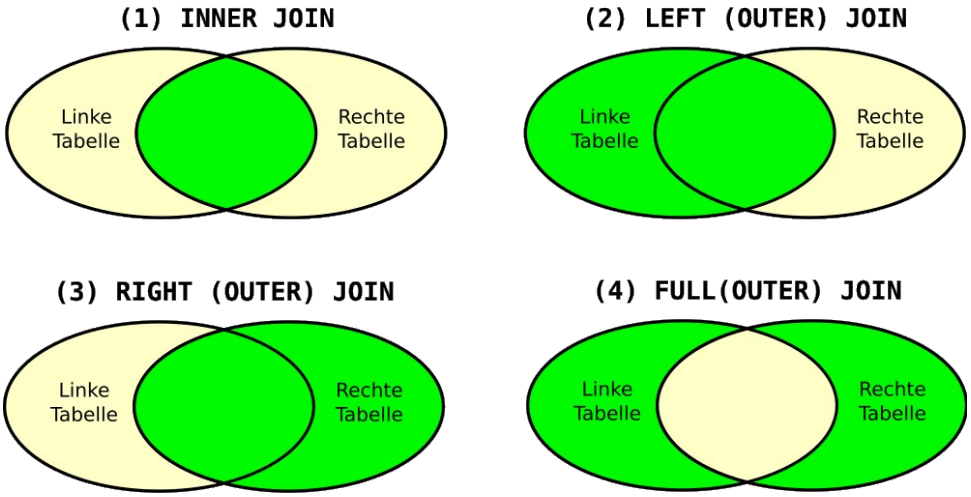
1.4 OUTER JOIN

In Aufgabe 11.4, Nr. 3 hatten Sie zu jedem Mitglied die Zahl seiner Foto-Uploads ermittelt. Sie haben dabei in der `JOIN`-Klausel etwas geschrieben wie `... ON users.id = photos.userid`. Damit bekommen Sie alle Datensätze, bei denen die Werte in den beiden Spalten übereinstimmen. Was aber ist mit einem Mitglied, dass keine Fotos hochgeladen hat? Da seine `id` in der Spalte `photos.userid` nicht auftaucht, wird es nicht in dem Ergebnis gelistet.

Wir haben also als Ergebnis eine Liste aller Mitglieder mit Foto-Uploads

erzeugt, aber nicht diejenigen Mitglieder die *keine* Foto-Uploads getätigt haben.

In der folgenden Abbildung sehen Sie die vier verschiedenen Formen des `JOINS`:



Vier `JOIN`-Arten

1. Beim `INNER JOIN` werden wie gesagt nur diejenigen Datensätze in das Zwischenergebnis übernommen , die einen passenden Datensatz *sowohl* in der linken als auch in der rechten Tabelle haben. In SQL kann das Wort `INNER` weggelassen werden.
2. Beim `LEFT OUTER JOIN` werden alle Datensätze in das Zwischenergebnis übernommen, die in der *linken* Tabelle enthalten sind. Wenn es passende Werte in der *rechten* Tabelle gibt, werden diese aufgenommen, ansonsten wird `NULL` eingetragen. In SQL kann das Wort `OUTER` weggelassen werden.
3. Beim `RIGHT OUTER JOIN` werden alle Datensätze in das Zwischenergebnis übernommen, die in der *rechten* Tabelle enthalten sind. Wenn es passende Werte in der *linken* Tabelle gibt, werden diese aufgenommen, ansonsten wird `NULL` eingetragen. In SQL kann das Wort `OUTER` weggelassen werden.
4. Der `FULL OUTER JOIN` hat in der Praxis nur ein geringe Bedeutung. Hier werden alle Datensätze in das Zwischenergebnis aufgenommen, die in nur in einer der beiden Tabellen stehen. Viele DBMS können keinen `FULL OUTER JOIN`, so auch das von InstaHub.

Auf das Ergebnis des `JOINS` werden dann die `WHERE`- und die `GROUP-BY`-Klausel angewendet.

SQL kennt außerdem noch den `NATURAL JOIN`. Dies ist ein `INNER JOIN`, bei dem das `ON` weggelassen wird und automatisch alle Attribute herangezogen werden, die denselben Namen haben. Der Befehl

```
SELECT  *
FROM    users NATURAL JOIN photos
```

ist identisch mit dem Befehl

```
SELECT  *
FROM    users INNER JOIN photos
        ON (users.created_at=photos.created_at AND user-
s.updated_at = photos.updated_at)
```

Wenn man mit `NATURAL JOIN` arbeiten will, muss man also bei der Benennung der Spaltennamen sehr aufpassen.

Aufgabe 11.10 Mehr Aufgaben mit JOIN

Entwickeln Sie jeweils eine SQL-Abfrage mit `JOIN`! (Tipp: Sie benötigen sowohl `INNER` als auch `OUTER JOIN`!)

1. Gesucht sind die Top 10 der am häufigsten gelikten Fotos!
2. Gesucht sind die Top 10 der am häufigsten kommentierten Fotos!
3. Gesucht sind die Top 10 der am häufigsten gefolgten User!
4. Gesucht sind die “Bottom 10”, also die am seltensten gelikten Fotos. Ihre Abfrage muss berücksichtigen, dass diese Liste sowohl völlig ungelikte als auch selten gelikte Fotos enthalten kann.
5. Gesucht sind die “Bottom 10”, also die am seltensten gefolgten Nutzer*innen, die nach dem 31.12.1990 geboren wurden. Ihre Abfrage muss berücksichtigen, dass diese Liste sowohl Personen ohne jeglichen Follower als auch selten Personen mit wenigen Followern enthalten kann.