GROUP 96 Ludwing Berglind Simon Severinsson

users(userID: integer, name: string)

friendships(userID1: integer, userID2: integer)

posts(postID: integer, userID: integer, title: string, date: date, place: string)

tags(postID: integer, tag: string)

image_posts(postID: integer, imgURL: string, filter: string)

text_posts(postID: integer, textContent: string)

video_posts(postID: integer, vidURL: string, codec: string)

likes(postID: integer, userID: integer, timestamp: timestamp)

events(eventID: integer, userID: integer, title: string, place; string, startDate: timestamp, endDate: timestamp)

attendees(eventID: integer, userID: integer)

subscription(subscriptionID: integer, userID: integer, dop: date, paymeth: string)

1)

a)

$$\text{NoLikes} \leftarrow \Upsilon_{postID,\ \text{COUNT}(userID) \rightarrow noLikes}\text{Likes}$$

$$\pi_{userID}\left(\sigma_{noPosts \geqslant 5}\left(\Upsilon_{userID,\ \text{COUNT}(postID) \rightarrow noPosts}\left(\text{Posts} \bowtie \left(\sigma_{noLikes \geqslant 10}\text{NoLikes}\right)\right)\right)\right)$$

First we create NoLikes which is the number of likes on each posts and call the count noLikes, which we get by counting all the userID's grouped by postID in Likes. We then select all rows with a noLikes of 10 or higher. We then join that select with Posts, which essentially leaves us with all entries in Posts that have 10 likes or more, count postID's in the join grouped by userID and call the result of COUNT noPosts, which is the number of posts with 10 or more likes each user has made. We then select all entries where noPosts is at least 5 and subsequently project userID onto that, which leaves us with the userID's of all users that have made 5 or more posts with 10 or more likes.

b)

$$\text{Friends} \leftarrow \Upsilon_{userID1,\ \text{COUNT}(userID2) \rightarrow noFriends}\left(\text{Friendships} \cup \rho_{(userID2,\ userID1)}\left(\text{Friendships}\right)\right)$$

$$\Upsilon_{\text{AVG}(noLikes) \rightarrow avg}\left(\Upsilon_{postID,\ \text{COUNT}(userID) \rightarrow noLikes}\left(\text{Likes} \bowtie \pi_{postID}\left(\text{Posts} \bowtie \left(\sigma_{noFriends > 50}\left(\text{Friends}\right)\right)\right)\right)\right)$$

$$\cup$$

$$\Upsilon_{\text{AVG}(noLikes) \rightarrow avg}\left(\Upsilon_{postID,\ \text{COUNT}(userID) \rightarrow noLikes}\left(\text{Likes} \bowtie \pi_{postID}\left(\text{Posts} \bowtie \left(\sigma_{noFriends \leqslant 50}\left(\text{Friends}\right)\right)\right)\right)\right)$$

We start by taking the union of Friendships with Friendships where userID1 is renamed to userID2 and vice versa, effectively 'mirroring' all friendships. If we dont do this each user in a friendship will only appear in one column of the relation, but in this union, if x is friends with y there will be one row in Friendships where userID1 = x, userID2 = y and one row where userID1 = y and userID2 = x. We then count userID2 grouped by userID1, which gives us the number of friends each user has, called noFriends, and assign the result of this to Friends. We then have a union where each side of it involves Friends. On the left-hand side we select

all entries from Friends where noFriends is greater than 50, join that with Posts and project postID which results in the postID's of all posts made by users with more than 50 friends. Join Likes with that result and then count the number of userID's grouped by postID in the join, resulting in the amount of likes on all those posts called noLikes and finally take the average of noLikes. The final result is just a number which is the average amount of likes on all posts made by users with more than 50 followers. On the right-hand side of the union we do the exact same thing, except we begin by selecting rows from Friends where noFriends is at most 50, so that the whole right-hand side results in the average number of likes on all posts made by users with at most 50 followers. So in the end we have a union of average amount of likes on posts by users with more than 50 followers and the average of those with at most 50 followers.

c)

$$\text{NoSubLikes} \leftarrow \gamma_{postID,\ \text{COUNT}(userID) \rightarrow noSubLikes}$$
$$\left(\text{Likes} \bowtie_{\text{Likes}.userID=subscription.userID \text{ AND Subscriptions}.date \leqslant Likes.timestamp \leqslant \text{Subscriptions}.dop+30} \text{Subscriptions}\right)$$
$$\text{NoLikes} \leftarrow \gamma_{postID,\ \text{COUNT}(userID) \rightarrow noLikes} \text{Likes}$$
$$\pi_{postID,\ (noSubLikes\ /\ noLikes) * 100 \rightarrow subscribedLikes\%} \left(\text{NoLikes} \bowtie \text{NoSubLikes}\right)$$

Here we start by creating NoSubLikes which is the amount of likes made on a post by users that were subsscribed at the time of liking. This is done by joining Likes with Subscriptions on the condition that that userID's are the same and that the date of the like is between (lower and upper inclusive) the date startDate (dop) and endDate (dop + 30) of the subscription. We then count userID's grouped by postID's, which in the end results in the number of 'subscribed likes' on every post. We also create NoLikes by taking the count of userID's grouped by postID on Likes. NoLikes and NoSubLikes are then joined, resulting in all posts and their respective noLikes and noSubLikes, we then do an extended projection of postID and (noSubLikes / noLikes) * 100 → subscribedLikes% which results in a list of all postID's and their respective percentage of likes that were made by subscribed users.

2)

a)

$\pi_{postID}\text{ImagePosts} \subseteq \pi_{postID}\text{Posts}$

Every postID in ImagePosts must refer to a postID in Posts, this constraint is here desbcribed by stating that the projection of postID onto ImagePosts must be a subset of the projection of postID onto Posts, meaning that there can be no postID in ImagePosts that is not also in Posts.

b)

$\sigma_{paymeth \neq 'klarna' \text{ AND } paymeth \neq 'swish' \text{ AND } paymeth \neq 'card' \text{ AND } paymeth \neq 'bitcoin'} \left(\text{Subcription}\right) = \varnothing$

In Sucscription the paymeth has to have one of the values 'klarna', 'swish', 'card' or 'bitcoin', this constraint is here illustrated by selecting all the rows from Subscription where paymeth is neither of those values and stating that the result of the selection must be empty, meaning that there can be no entries that do not have one of these values in the paymeth column.

c)

$$\sigma_{\text{U1}.userID=\text{U2}.userID \text{ AND U1}.name\neq\text{U2}.name}\left(\rho_{\text{U1}}\left(\text{Users}\right) \times \rho_{\text{U2}}\left(\text{Users}\right)\right) = \varnothing$$

In Users, userID is a primary key, meaning that every row's userID has to be unique amongst all others. This constraint is here illustrated by taking the cartesian product of Users named U1 and Users named U2, then stating that selection of rows where the userID's match but the names do not, results in an empty set. This works beause all userID's are unique and connected to one name, there can be no entries in the cartesian product where userID's match but names do not.