BlueBlueBlueBlueBlueBlueBlueBlue
There are two tables which store friend requests: fct_request, which contains all sent requests and fct_accept, which contains all requests which have been accepted. These tables look something like:

fct_request

```
+------------+------------+---------------+
| dateid     | sender_uid | recipient_uid |
+------------+------------+---------------+
| 2015-01-01 |          1 |             2 |
| 2015-01-01 |          1 |             4 |
| 2015-01-02 |          3 |             1 |
| 2015-01-03 |          4 |             5 |
+------------+------------+---------------+
```

fct_accept

```
+------------+--------------+------------+
| dateid     | accepter_uid | sender_uid |
+------------+--------------+------------+
| 2015-01-01 |            2 |          1 |
| 2015-01-02 |            4 |          1 |
| 2015-01-02 |            1 |          3 |
+------------+--------------+------------+
```

A third utility table, just_date, contains all dates that we care about

```
+------------+|
dateid       |
+------------+
| 2015-01-01 |
| 2015-01-02 |
| 2015-01-02 |
+------------+
```

Some rules:
1. UIDs are unique user identification numbers.
2. You can only send a friend request to a person one time.
3. You cannot send a friend request to someone who has already sent you a friend request.
4. You cannot remove friends once added

Q1: What percent of friend requests are accepted?

The simplest way is to just count up the # of rows in each table and divide by each other (SQL is surprisingly inconvenient to do this in). This would look like

```
count(*)
from fct_request
```

and

```
count(*)
from fct_accept
```

and then take these values and divide them offline.

A messier solution involves a left join:

```
select count(*) / sum(case when acc.dateid is not null then 1 else 0 end)
from fct_request rec
left join fct_accept acc
on rec.sender_uid = acc.sender_uid
and rec.recipient_uid = acc.accepter_id
```

This relies on the fact that a left join preserves all rows in the first table but only those in the second table which match.

Q1.5: What percent of all previous friend requests are accepted, cumulatively, by date?

The easy way to do this is to use the just_date utility table. Getting the # sent by a particular date looks like:

```
select jd.dateid, count(*) as total_sent
from just_date jd
join fct_request rec
on jd.dateid >= rec.dateid
group by 1
```

Our final query involves doing this twice and joining together.

```
select x.dateid, total_accepted / total_sent as pct_yes
from (
select jd.dateid, count(*) as total_sent
from just_date jd
join fct_request rec
on jd.dateid >= rec.dateid
group by 1
) x
join (
select jd.dateid, count(*) as total_accepted
from just_date jd
join fct_accept rec
on jd.dateid >= rec.dateid
group by 1
) y
on x.dateid = y.dateid
```

Long, but not terribly complicated

Q2: Who has the most friends?

This question is tricky. We want to combine the number of accepted outgoing friend requests and the number of accepted incoming friend requests by uid and then add them together.

We can get each of these easily enough

```
select accepter_uid, count(*) as inbound
from fct_accept
group by accepter_uid
```

AND

```
select sender_uid, count(*) as outbound
from fct_accept
group by sender_uid
```

However, to combine them, we'll have to use the almost-never-seen-in-the-wild outer join. The following query will get your answer:

```
select accepter_uid, sender_uid, coalesce(inbound, 0) + coalesce(outbound, 0) as total_friends
from (
select accepter_uid, count(*) as inbound
from fct_accept
group by accepter_uid
) x
outer join (
select sender_uid, count(*) as outbound
from fct_accept
group by sender_uid
) y
on x.accepter_uid = y.sender_uid
order by total_friends desc
limit 1
```

If you REALLY want to be fancy you can replace the first line with something like:
case when accepter_uid is null then sender_uid else accepter_uid end as uid,
inbound+outbound as total_friends

but any reasonable human can read the answer. The limit of 1 is also unnecessary--just order and take the row at the top.

(Alternate: Another way to combine the two rows is with a UNION. This also eliminates the need for an outer join.)

```
SELECT uid, COUNT(*) AS num_friends
  FROM (SELECT accepter_uid AS uid FROM fct_accept)
       UNION ALL
       (SELECT sender_uid AS uid FROM fct_accept)
 GROUP BY uid
 ORDER BY num_friends DESC
 LIMIT 1
```