Given two tables, message and users:

```
create temporary table messages (
    id integer,
    user_id integer,
    message_body varchar(4096),
    created_at timestamp without time zone,
    sent_at timestamp without time zone,
    was_scheduled boolean,
    recipient_count integer,
    attachment_type enum('file', 'payment', 'sticker'),
    locale varchar(255),
    message_type enum('chat', 'broadcast', 'group_chat')
);

create temporary table users (
    user_id integer,
    email varchar(1024),
    created_at timestamp,
    deleted_at timestamp,
    country varchar(255),
    state varchar(255),
    roles varchar(255),
    birthdate date
);
```

Between August 1st, 2014 at 9am PST and December 1st, 2014, at 10 pm PST, how many group chat messages were sent by parents in California?

```
select
    count(m.id)
from
    messages m
join
    users u
on
    m.user_id = u.user_id
where
    m.sent_at between '2014-08-01 09:00:00'::timestamp with time zone at time zone 'PST' and
    and
        u.roles like '%parents%'
    and
        u.state like '%CA%'
    and
        u.country like '%USA%'
    and
        m.message_type like '%group_chat%'
```

```
;
```

List the number of stickers sent per day between January 1st, 2014 to April 30th, 2014 by users who have never deleted their accounts.

```
select
    date_trunc('day', m.sent_at),
    count(m.id)
from
    messages m
join
    users u
on
    m.user_id = u.user_id
where
    m.sent_at between '2014-01-01 00:00:00'::timestamp with time zone at time zone 'PST' anc
    and
        u.deleted_at is null
    and
        m.attachment_type like '%sticker%'
group by
    date_trunc('day', m.sent_at)
;
```

We define a teacher as “activated” if they send a message to at least 3 people within 3 days of signing up for Remind. Calculate the daily activation rate for teachers who signed up for the service since January 1st, 2014 to the present day. This answer should have two columns and should take the form “Date”, and “Activation Rate”.

```
select
    date_trunc(day, u.created_at) `Date`,
    sum(case when count(m.user_id) >= 3 then 1 else 0 end) / count(distinct m.user_id) `Acti
from
    messages m
join
    users u
on
    m.user_id = u.user_id
where
    u.created_at > '2014-01-01 00:00:00'::timestamp with time zone at time zone 'PST'
    and
        u.roles like '%teacher%'
    and
        m.sent_at between u.created_at::timestamp with time zone at time zone 'PST' and u.cr
group by
        date_trunc(day, u.created_at)
;
```

Given the website table:

```
create temporary table website (
    category varchar(255),
    website enum(
        'www.metis.com'
        'www.datacamp.com',
        'www.facebook.com',
        'www.google.com'
    )
);
```

Find counts by category and url that are greater than 1.

```
select
    *
from (
    select
        w.category,
        w.url,
        count(*) visit_count
    from
        website w
    group by
        w.category,
        w.url
) t
where
    t.visit_count > 1
order by
    t.visit_count desc
;
```

or

```
select
    w.category,
    w.url,
    count(w.category) visit_count
from
    website w
group by
    w.category,
    w.url
having
    count(w.category) > 1
;
```

Find counts by category and url greater than 1 and url displayed with null if category and url counts are not greater than 1.

```
select
    t.category,
    case when t.visit_count > 1 then t.url else null end url,
    sum(t.visit_count) visit_count
from (
    select
        w.category,
        w.url,
        count(w.category) visit_count
    from
        website w
    group by
        w.category,
        w.url
    having
        count(w.category) > 1
) t
group by
    t.category,
    case when t.visit_count > 1 then t.url end
;
```

Given the friend_request table and request_accepted table:

```
create temporary table friend_request (
    requestor_id,
    sent_to_id,
    time
);

create temporary table request_accepted (
    acceptor_id
    requestor_id
    time
);
```

Find the overall acceptance rate of requests. It will help to define what acceptance means, and what acceptance rate means

```
select
    sum(case when a.category = 'Accepted' then 1 else 0 end) / count(a.category)
from (
    select
        ra.acceptor_id,
        fr.requestor_id,
        case
            when datediff('day', ra.time accepted_time, fr.time requested_time) >= 7
            then 'Rejected'
            else 'Accepted'
        end category
    from
        request_accepted ra
    inner join
        friend_request fr
    on
        ra.requestor_id = fr.requestor_id and ra.sent_to_id = fr.acceptor_id
) a
;
```

```
from pyspark.sql import *
from pyspark.sql.functions import *
from pyspark.sql.types import *
from datetime import datetime as dt

requestClass = udf(lambda tdiff: 'Rejected' if tdiff >= 7 else 'Accepted')

acceptance_rate = (
    request_accepted
    .join(
        friend_request
        .withColumnRenamed('time', 'time_requested')
        [
            friend_request.requestor_id == request_accepted.requestor_id
            friend_request.sent_to_id == request_accepted.acceptor_id
        ]
        'inner'
    )
    # define how long a pending request is considered rejected
    .withColumn(
        'status',
        requestClass(
            (unix_timestamp(col('time') - unix_timestamp(col('time_requested'))))/86400
        )
    )
    .withColumn(
        'accept_binary',
        when(col('status') == 'Accepted', 1).otherwise(0)
    )
    .agg(
        (count(col('accept_binary')) / count(col('status'))).alias('acceptance_rate')
    )
)
```

Which user gained the most new friends on a particular date, ‘2017-08-01’?

```
create table if not exists acceptance_status
as
select
    ra.acceptor_id,
    fr.requestor_id,
    case when a.category = 'Accepted' then 1 else 0 end status
from (
    select
        ra.acceptor_id,
        fr.requestor_id,
        case
            when datediff('day', ra.time accepted_time, fr.time requested_time) >= 7
            then 'Rejected'
            else 'Accepted'
        end category
    from
        request_accepted ra
    inner join
        friend_request fr
    on
        ra.requestor_id = fr.requestor_id and ra.sent_to_id = fr.acceptor_id
    where
        ra.time = '2017-08-01'


select
    d.id,
    sum(d.count)
from (
    select
        b.acceptor_id id
        count(b.acceptor_id) count
    from
        acceptance_status b
    where
        b.status = 1
    group by
        b.acceptor_id

    union all

    select
```

```
        c.requestor_id id
        count(c.acceptor_id) count
    from
        acceptance_status c
    where
        c.status = 1
    group by
        c.requestor_id
) d
group by
    d.id
;

friend_df = (
    request_accepted
    .join(
        friend_request
        .withColumnRenamed('time', 'time_requested')
        [
            friend_request.requestor_id == request_accepted.requestor_id
            friend_request.sent_to_id == request_accepted.acceptor_id
        ]
        'inner'
    )
    .where(col('time_requested') == '2017-08-01')
    .withColumn(
        'accept_binary',
        when(col('status') == 'Accepted', 1).otherwise(0)
    )
    .where(col('accept_binary') == 1)
)

friends = (
    friend_df
    .groupBy('requestor_id')
    .agg(sum(col('accept_binary')).alias('count'))
    .withColumnRenamed('requestor_id', 'id')
    .unionAll(
        friend_df
        .groupBy('acceptor_id')
        .agg(sum(col('accept_binary')).alias('count'))
        .withColumnRenamed('acceptor_id', 'id')
    )
    .groupBy('id')
    .agg(sum(col('count')).alias('count'))
    .sort(col('count').desc())
```

```
)
```

Given the friends and likes tables:

```
create temporary table friends (
    user_id int not null,
    friend_id int not null
);

create temporary table likes (
    user_id int not null,
    page_id int not null
);
```

Write an SQL query that makes recommendations using the pages that your friends liked. It should not recommend pages you already like.

```
select
    f.user_id,
    l.page_id
from
    friends f
join
    likes l on f.friend_id = l.user_id
where
    f.user_id != l.user_id
    and
        l.page_id is not null
;
```

```
from pyspark.sql import *
from pyspark.sql.functions import *
from pyspark.sql.types import *
from datetime import datetime as dt

recommended = (
    friends
    .join(
        likes
        .withColumnRenamed('user_id', 'user_id_l'),
        [friends.friend_id == likes.user_id],
        'inner'
    )
    .where(
        (col('user_id_likes') != 'your_own_id') &
        (col('user_id') != col('user_id_l'))
        (col('page_id').isNotNull())
    )
    .select('user_id', 'page_id')
)
```

Given the student_log and demographics tables:

```
create temporary table student_log (
    date,
    student_id,
    attendance
);

create temporary table demographics (
    student_id,
    school_id,
    grade_level,
    date_of_birth,
    hometown
);
```

What was the overall attendance rate for the school district yesterday?

```
select
    sum(case when sl.attendance = True then 1 else 0) present,
    count(distinct sl.student_id) total
from
    student_log sl
left join
    demographics d
on
    sl.student_id = d.student_id
where
    d.school_id in ('example_id_1', 'example_id_2')
    and
    sl.date = today() - interval '1 day'
;
```

```
from pyspark.sql import *
from pyspark.sql.functions import *
from pyspark.sql.types import *
from datetime import datetime as dt

attendance_rate = (
    student_log
    .join(demographics, 'student_id', 'left')
    .where(
        (col('school_id').isin(['school_ids_in_district'])) &
        (
            col('date') == date(unix_timestamp(date(dt.datetime.today())) - 86400)
        )
    )
    .withColumn(
        'attendance_binary', when(col('attendance') == True, 1).otherwise(0)
    )
    .agg(
        sum(col('attendance_binary').alias('present')),
        count(col('student_id').alias('total'))
    )
    .withColumn('attendance_rate', col('attendance') / col('total'))
)
```

Which grade level currently has the most students in this school district?

```
select
    d.grade_level,
    count(distinct d.student_id) unique_students
from
    demographics d
where
    d.school_id in ('example_id_1', 'example_id_2')
group by
    d.grade_level
;
```

```
student_count = (
    demographics
    .where(col('school_id').isin(['school_ids_in_district']))
    .groupBy('grade_level')
    .agg(countDistinct(col('student_id').alias('unique_students')))
)
```

Which school had the highest attendance rate? The lowest?

```
select
    a.school_id,
    a.present / a.total attendance_rate
from (
    select
        d.school_id,
        sum(case when sl.attendance = True then 1 else 0) present,
        count(distinct sl.student_id) total
    from
        student_log sl
    left join
        demographics d
    on
        sl.student_id = d.student_id
    where
        d.school_id in ('example_id_1', 'example_id_2')
    group by
        d.school_id
) a
;

attendance_rate = (
    student_log
    .join(demographics, 'student_id', 'left')
    .where(col('school_id').isin(['school_ids_in_district']))
    .withColumn(
        'attendance_binary', when(col('attendance') == True, 1).otherwise(0)
    )
    .groupBy('date', 'school_id')
    .agg(
        sum(col('attendance_binary').alias('present')),
        count(col('student_id').alias('total'))
    )
    .withColumn('attendance_rate', col('attendance') / col('total'))
    # aggregate up to average?
)
```

Given a lifetime_music_actions table (as of yesterday and of all time) and music_actions table (as of today):

```
create temporary table lifetime_music_actions (
    dt datetime,
    user_id bigint,
    song_id bigint,
    cnt int
);

create temporary table music_actions (
    dt datetime,
    timestamp bigint,
    user_id bigint,
    song_id bigint
);
```

Consider a table lifetime_music_actions that for any dt value that has all the user song pairs that have ever existed and the count of listens for each pair. Write a query to update that table for today given that you have it for yesterday and music actions for today.

```
select
    a.dt,
    a.userid,
    a.song,
    sum(a.count) count
from (
    select
        lma.dt
        lma.userid,
        lma.song,
        lma.count
    from
        lifetime_music_actions lma
    union all
    select
        ma.dt,
        ma.userid,
        ma.song,
        count(ma.timestamp) count
    from
        music_actions ma
    where
        to_date(ma.dt) = today()
    group by
        ma.dt
        ma.userid,
        ma.song
) a
group by
    a.dt
    a.userid,
    a.song
;
```

```
from pyspark.sql import *
from pyspark.sql.functions import *
from pyspark.sql.types import *
from datetime import datetime as dt

updated = (
    lifetime_music_actions
    .select(
        col('dt'),
        col('userid'),
        col('song'),
        col('cnt').alias('count')
    )
    .unionAll(
        music_actions
        .where(col('dt') == dt.datetime.today())
        .groupBy(
            col('dt'),
            col('userid'),
            col('song')
        )
        .agg(count(col('timestamp')).alias('count'))
    )
)
```

There is a table that tracks every time a user turns a feature on or off:

```
create temporary table tracking (
    user_id,
    action enum('on', 'off'),
    date,
    time
);
```

How many users turned the feature on today?

```
select
    count(distinct t.user_id)
from
    tracking t
where
    t.date = today()
    and
        t.action = 'on'
;
```

How many users have ever turned the feature on?

```
select
    count(distinct t.user_id)
from
    tracking t
group by
    t.action
;
```

In a table that tracks the status of every user every day, how would you add today's data to it?

```
select
```

```
    a.date,
    sum(a.count)
from (
    select
        ta.date,
        ta.user_id,
        ta.count
    from
        tracking_all ta
    union all
    select
        t.date
        count(distinct t.user_id)
    from
        tracking t
    where
        t.date = today()
        and
            t.action = 'on'
    group by
        t.date
) a
group by
    a.date
;
```

```
w = Window.partitionBy('date', 'user_id').orderBy(col('time').desc())

updated = (
    tracking
    .withColumn('rank', rank().over(w))
    .where(col('rank') == 1)
    .drop(col('rank'))
    .select('date', 'user_id', 'action')
    .unionAll(
        tracking
        .withColumn('rank', rank().over(w))
        .where(col('rank') == 1)
        .drop(col('rank'))
        .select('date', 'user_id', 'action')
    )
)
```

Given the forum_topics table and posts table:

```
create temporary table forum_topics (
    id,
    forum_cateogry_id
);

create temporary table posts (
    id,
    object_id,
    object_type enum('forum_topic', 'forum', 'user', 'post'),
    last_replied datetime not null,
    status enum('approved', 'unapproved', 'disabled')
);
```

Get a count of all forum_topics and replies in posts table associated with a forum topic in the forum_topics table.

```
select
    ft.forum_category_id,
    count(distinct p1.id) forum_topics_count,
    count(p2.id) replies_count
from
    forum_topics ft
left join
    posts p1
on
    ft.id = p1.object_id
    and
        p1.object_type = 'forum_topic'
    and
        p1.status = 'approved'
left join
    posts p2
on
    p1.id = p2.object_id
    and
        p2.object_type = 'post'
    and
        p2.status = 'approved'
where
    ft.forum_category_id in ('example_id_1', 'example_id_2')
group by
    ft.forum_category_id
```

```
from pyspark.sql import *
from pyspark.sql.functions import *
from pyspark.sql.types import *
from datetime import datetime as dt
```

```
counts_table = (
    forum_topics
    .join(
        posts
        .where(
            (col('object_type') == 'forum_topic') &
            (col('status') == 'approved')
        )
        .as('posts_1'),
        [forum_topics.id == posts_1.object_id],
        'left'
    )
    join(
        posts
        .where(
            (col('object_type') == 'post') &
            (col('status') == 'approved') &
            (col('last_replied').isNotNull())
        )
        .as('posts_2'),
        [posts_1.id == posts_2.object_id],
        'left'
    )
    .withColumn(
        'approved_forum_topic',
        when(
            (posts_1.object_type == 'forum_topic') &
            (posts_1.status == 'approved'), 1
        ).otherwise(0))
    .withColumn(
        'replied_post',
        when(
            (posts_2.object_type == 'post') &
            (posts_2.status == 1), 1
        ).otherwise(0))
    .groupBy('forum_category_id')
    .agg(
        count(col('approved_forum_topic')).alias('forum_topics_count'),
        count(col('replied_post')).alias('replies_count')
    )
)
```