SUPABASE DATABASE

A comprehensive description of the entire database.
The specific requests are highlighted in purple.

The following database is designed for a mobile application developed using FlutterFlow.

Since I am not an official developer, any advice is welcome and appreciated.

Tables:

1. questions_table
2. question_topics
3. answers_table
4. users_questions_relations
5. users_answers
6. users_answers_friends
7. users_friends
8. users_details
9. nationalities_table
10. colors_table
11. all_users_statistics
12. users_notifications

Views:

1. all_users_statistics_view
2. answers_list_view
3. colors_list_view
4. question_list_view
5. users__details_list_view
6. user_friends_view
7. users_notifications_view
8. users_statistics_view
9. user_friends_answers_view

TABLES:

[1)questions_table]

code:

```
create table
  public.questions_table (
    id bigint generated by default as identity,
    created_at timestamp with time zone not null default now(),
    description text not null,
```

```
  voti bigint null default '0'::bigint,
  class text not null,
  color_code text not null,
  constraint Questions_pkey primary key (id),
  constraint public_questions_table_color_fkey foreign key (color_code) references colors_table (color_code)
on update cascade on delete set null,
  constraint questions_table_color_check check ((length(color_code) < 500))
) tablespace pg_default;
```

Comment:

The table contains the details of each question.
The table has a column called "voti" (votes) that represents the number of people who have answered this question (trigger (RispostaAggiunta) is activated when a new row is added to the users_answers table).
Is it correct to have this data here?
Could I calculate it and have it only in the "question_list_view" view?
What is the best way to do it?

[2)question_topics]

code:

```
create table
 public.question_topics (
  id bigint generated by default as identity,
  created_at timestamp with time zone not null default now(),
  topic_one text not null,
  topic2 text null,
  topic3 text null,
  topic4 text null,
  question_id bigint not null,
  constraint question_topics_pkey primary key (id),
  constraint question_topics_question_id_key unique (question_id),
  constraint public_question_topics_question_id_fkey foreign key (question_id) references questions_table
(id) on delete cascade,
  constraint public_question_topics_topic_one_fkey foreign key (topic_one) references colors_table (topic)
) tablespace pg_default;
```

Comment:

The table contains the topics covered by each question.

[3)answers_table]

code:

```
create table
```

```
  public.answers_table (
    id bigint generated by default as identity,
    created_at timestamp with time zone not null default now(),
    question_id bigint not null,
    description text not null,
    lettera text not null,
    voti_tot bigint not null default '0'::bigint,
    voti_female bigint not null default '0'::bigint,
    voti_male bigint not null default '0'::bigint,
    voti_teen bigint not null default '0'::bigint,
    voti_under40 bigint not null default '0'::bigint,
    voti_over40 bigint not null default '0'::bigint,
    voti_italiani bigint not null default '0'::bigint,
    voti_inglesi bigint not null default '0'::bigint,
    voti_americani bigint not null default '0'::bigint,
    voti_altre_nazionalità bigint not null default '0'::bigint,
    voti_under30 bigint not null default '0'::bigint,
    constraint Answers_pkey primary key (id),
    constraint answers_table_id_key unique (id),
    constraint unique_answer_letter_description unique (lettera, description, question_id),
    constraint unique_answer_question_description unique (question_id, description),
    constraint public_answers_table_question_id_fkey foreign key (question_id) references questions_table (id)
on delete cascade
  ) tablespace pg_default;
```

Comment:

Each row in the table is an answer to a question.
Here again, I ask myself the same question: **Is it correct to put the vote data here, or is it better to calculate it in the "answers_list_view"?**
The votes are updated by the trigger in the "users_answers" table (RispostaAggiunta)

[4)users_questions_relations]

code:

```
create table
  public.users_questions_relations (
    id bigint generated by default as identity,
    created_at timestamp with time zone not null default now(),
    done boolean not null default false,
    prefered boolean not null default false,
    to_public boolean not null default false,
    user_id uuid not null,
    question_id bigint not null,
    to_public_all boolean not null default false,
    constraint users_questions_pkey primary key (id),
```

```
  constraint unique_userid_questionid unique (user_id, question_id),
  constraint public_users_questions_relations_question_id_fkey foreign key (question_id) references
questions_table (id) on delete cascade,
  constraint public_users_questions_user_id_fkey foreign key (user_id) references auth.users (id)
) tablespace pg_default;
```

Comment:

Each row in the column represents the relationship between user and question.
A row is added when the trigger (RispostaAggiunta) is activated.

[5)users_friends]

code:

```
create table
  public.users_friends (
    id bigint generated by default as identity,
    created_at timestamp with time zone not null default now(),
    user1_id uuid not null,
    user2_id uuid not null,
    constraint users_friends_pkey primary key (id),
    constraint users_friends_user1_id_user2_id_key unique (user1_id, user2_id),
    constraint public_users_friends_user1_id_fkey foreign key (user1_id) references auth.users (id),
    constraint public_users_friends_user2_id_fkey foreign key (user2_id) references auth.users (id)
  ) tablespace pg_default;
```

Comment:

Each row in the column represents a friendship relationship between 2 users.
1) I would like to add the condition that the pair (user1,user2)=(user2,user1), to avoid duplicates in the table.

[6)users_answers]

code:

```
create table
  public.users_answers (
    id bigint generated by default as identity,
    created_at timestamp with time zone not null default now(),
    my_answer_id bigint not null,
    user_id uuid not null,
    mass_answer_id bigint not null,
    question_id bigint not null,
```

```
  constraint Users_Answers_pkey primary key (id),
  constraint unique_user_question unique (user_id, question_id),
  constraint public_Users_Answers_answer_id_fkey foreign key (my_answer_id) references answers_table (id),
  constraint public_users_answers_mass_answer_id_fkey foreign key (mass_answer_id) references answers_table
(id),
  constraint public_users_answers_question_id_fkey foreign key (question_id) references questions_table (id)
on delete cascade,
  constraint public_users_answers_user_id_fkey foreign key (user_id) references auth.users (id) on update
cascade on delete cascade
) tablespace pg_default;

create trigger "RispostaAggiunta"
after insert on users_answers for each row
execute function "RispostaAggiunta" ();

create trigger "RispostaCambiata"
after
update on users_answers for each row
execute function "RispostaCambiata" ();
```

Comment:

Each row in the table represents an answer to a user's question.

[7)users_details]

code:

```
create table
  public.users_details (
    id bigint generated by default as identity,
    created_at timestamp with time zone not null default now(),
    user_id uuid not null,
    born_date date not null,
    prefered_color text not null,
    gender text not null,
    name text not null,
    nickname text not null,
    profile_picture text null,
    nationality text not null,
    status text not null,
    constraint users_details_pkey primary key (id),
    constraint users_details_nickname_key unique (nickname),
    constraint users_details_user_id_key unique (user_id),
    constraint public_users_details_nationality_fkey foreign key (nationality) references nationalities_table
(nationality),
    constraint public_users_details_user_id_fkey foreign key (user_id) references auth.users (id) on update
cascade on delete cascade,
    constraint users_details_name_check check ((length(name) > 2))
  ) tablespace pg_default;


create trigger "UserDetailAggiunto"
after insert on users_details for each row
execute function "UserDetailAggiunto" ();
```

Comment:

Each row in the table represents the details of a user.

[8)users_notifications]

code:

```
create table
  public.users_notifications (
    id bigint generated by default as identity,
    created_at timestamp with time zone not null default now(),
    send_by uuid not null,
    send_to uuid not null,
    notification_type text not null,
    constraint users_notifications_pkey primary key (id),
    constraint users_notifications_send_by_send_to_notification_type_key unique (send_by, send_to,
notification_type),
    constraint public_users_notifications_send_by_fkey foreign key (send_by) references auth.users (id),
    constraint public_users_notifications_send_to_fkey foreign key (send_to) references auth.users (id)
  ) tablespace pg_default;
```

Comment:

Each row of the table represents a notification.
2) I would like the trio (user1,user2,"Friend Request")=(user2,user1,"Friend Request") to avoid duplicates in the table.

[9)users_answers_friends]

code:

```
create table
  public.users_answers_friends (
    id bigint generated by default as identity,
    created_at timestamp with time zone not null default now(),
    question_id bigint not null,
    user1_id uuid not null,
    user2_id uuid not null,
    user1_guess_answer_id bigint not null,
    user2_answer_id bigint null,
    user1_answer_id bigint not null,
    user2_guess_answer_id bigint null,
    constraint users_answers_friends_pkey primary key (id),
```

```
   constraint public_users_answers_friends_user1_answer_id_fkey foreign key (user1_answer_id) references
answers_table (id),
   constraint public_users_answers_friends_user1_guess_answer_id_fkey foreign key (user1_guess_answer_id)
references answers_table (id),
   constraint public_users_answers_friends_question_id_fkey foreign key (question_id) references
questions_table (id),
   constraint public_users_answers_friends_user2_guess_answer_id_fkey foreign key (user2_guess_answer_id)
references answers_table (id),
   constraint public_users_answers_friends_user2_id_fkey foreign key (user2_id) references auth.users (id),
   constraint public_users_answers_friends_user1_id_fkey foreign key (user1_id) references auth.users (id)
 ) tablespace pg_default;

create trigger "RispostaAmicoAggiunta"
after insert on users_answers_friends for each row
execute function "RispostaAmicoAggiunta" ();
```

Comment:

Each row in the table represents a relationship between 2 users and a question.

user2_id is the one who posted the question.
user1_id is the one who answered the question.
user1_guess_answer_id is the answer ID (answers_table.id) that user1_id thinks user2_id gave.
user2_guess_answer_id is the answer ID (answers_table.id) that user2_id thinks user1_id gave.
user2_answer_id is the actual answer of user2_id in users_answers.
user1_answer_id is the actual answer of user1_id in users_answers.
Is it correct to put user1_answer_id and user2_answer_id in this column even if the data is already present in the users_answers table?

Inserting a new row into this table activates the "RispostaAmicoAggiunta" trigger.

[10)nationalities_table]

code:

```
create table
 public.nationalities_table (
  id bigint generated by default as identity,
  created_at timestamp with time zone not null default now(),
  nationality text not null,
  constraint nationalities_table_pkey primary key (id),
  constraint nationalities_table_nationality_key unique (nationality)
 ) tablespace pg_default;
```

Comment:

Table with the nationalities.

[11)colors_table]

code:

```
create table
  public.colors_table (
    id bigint generated by default as identity,
    created_at timestamp with time zone not null default now(),
    color_text text not null,
    color_code text not null,
    topic text null,
    voti_tot bigint not null default '0'::bigint,
    constraint colors_table_pkey primary key (id),
    constraint colors_table_ # _key unique (color_code),
    constraint colors_table_color_text_key unique (color_text),
    constraint colors_table_topic_key unique (topic)
  ) tablespace pg_default;
```

Comment:

Table with colors associated with topic_one.

[12)all_users_statistics]

code:

```
create table
  public.all_users_statistics (
    id bigint generated by default as identity,
    created_at timestamp with time zone not null default now(),
    n_utenti bigint not null default '0'::bigint,
    n_utenti_f bigint not null default '0'::bigint,
    n_utenti_m bigint not null default '0'::bigint,
    n_utenti_teen bigint not null default '0'::bigint,
    n_utenti_under40 bigint not null default '0'::bigint,
    n_utenti_over40 bigint not null default '0'::bigint,
    n_utenti_italiani bigint not null default '0'::bigint,
    n_utenti_americani bigint not null default '0'::bigint,
    n_utenti_inglesi bigint not null default '0'::bigint,
    n_utenti_altra_nazionalità bigint not null default '0'::bigint,
    n_utenti_under30 bigint not null default '0'::bigint,
```

```
  constraint users_statistics_pkey primary key (id)
) tablespace pg_default;
```

Comment:

Table with statistics on all users of the platform. When a new row is inserted into "users_details", the trigger starts to update the values of this table.

Does it make sense to create a table like this with only one row, or is it better to create a view with these values?

VIEWS:

[1)all_users_statistics_view]

code:

```
create view
  public.all_users_statistics_view as
with
  age_groups as (
    select
      users_details.id,
      users_details.created_at,
      users_details.user_id,
      users_details.born_date,
      users_details.prefered_color,
      users_details.gender,
      users_details.name,
      users_details.nickname,
      users_details.profile_picture,
      users_details.nationality,
      case
        when users_details.born_date >= (current_date - '20 years'::interval) then 'Teen'::text
        when users_details.born_date >= (current_date - '30 years'::interval)
        and users_details.born_date < (current_date - '20 years'::interval) then 'Under30'::text
        when users_details.born_date >= (current_date - '40 years'::interval)
        and users_details.born_date < (current_date - '30 years'::interval) then 'Under40'::text
        else 'Over40'::text
      end as age_group
    from
      users_details
```

```
  )
select
  count(*) as n_utenti,
  count(
    case
      when age_groups.gender = 'Femmina'::text then 1
      else null::integer
    end
  ) as n_utenti_f,
  count(
    case
      when age_groups.gender = 'Maschio'::text then 1
      else null::integer
    end
  ) as n_utenti_m,
  count(
    case
      when age_groups.age_group = 'Teen'::text then 1
      else null::integer
    end
  ) as n_utenti_teen,
  count(
    case
      when age_groups.age_group = 'Under30'::text then 1
      else null::integer
    end
  ) as n_utenti_under30,
  count(
    case
      when age_groups.age_group = 'Under40'::text then 1
      else null::integer
    end
  ) as n_utenti_under40,
  count(
    case
      when age_groups.age_group = 'Over40'::text then 1
      else null::integer
    end
  ) as n_utenti_over40,
  case
    when count(*) > 0 then (
      count(
        case
          when age_groups.gender = 'Femmina'::text then 1
          else null::integer
        end
      )::double precision / count(*)::double precision * 100::double precision
    )::integer
    else 0
  end as utenti_f_perc,
```

```
case
  when count(*) > 0 then (
    count(
      case
        when age_groups.gender = 'Maschio'::text then 1
        else null::integer
      end
    )::double precision / count(*)::double precision * 100::double precision
  )::integer
  else 0
end as utenti_m_perc,
case
  when count(*) > 0 then (
    count(
      case
        when age_groups.age_group = 'Teen'::text then 1
        else null::integer
      end
    )::double precision / count(*)::double precision * 100::double precision
  )::integer
  else 0
end as utenti_teen_perc,
case
  when count(*) > 0 then (
    count(
      case
        when age_groups.age_group = 'Under30'::text then 1
        else null::integer
      end
    )::double precision / count(*)::double precision * 100::double precision
  )::integer
  else 0
end as utenti_under30_perc,
case
  when count(*) > 0 then (
    count(
      case
        when age_groups.age_group = 'Under40'::text then 1
        else null::integer
      end
    )::double precision / count(*)::double precision * 100::double precision
  )::integer
  else 0
end as utenti_under40_perc,
case
  when count(*) > 0 then (
    count(
      case
        when age_groups.age_group = 'Over40'::text then 1
        else null::integer
```

```
      end
    )::double precision / count(*)::double precision * 100::double precision
  )::integer
  else 0
end as utenti_over40_perc,
count(
  case
    when age_groups.nationality = 'Italia'::text then 1
    else null::integer
  end
) as n_utenti_italiani,
count(
  case
    when age_groups.nationality = 'England'::text then 1
    else null::integer
  end
) as n_utenti_inglesi,
count(
  case
    when age_groups.nationality = 'Usa'::text then 1
    else null::integer
  end
) as n_utenti_amaericani,
count(
  case
    when age_groups.nationality = 'Altra'::text then 1
    else null::integer
  end
) as "n_utenti_altra_nazionalità",
case
  when count(*) > 0 then (
    count(
      case
        when age_groups.nationality = 'Italia'::text then 1
        else null::integer
      end
    )::double precision / count(*)::double precision * 100::double precision
  )::integer
  else 0
end as utenti_italiani_perc,
case
  when count(*) > 0 then (
    count(
      case
        when age_groups.nationality = 'Usa'::text then 1
        else null::integer
      end
    )::double precision / count(*)::double precision * 100::double precision
  )::integer
  else 0
```

```
  end as utenti_americani_perc,
  case
    when count(*) > 0 then (
      count(
        case
          when age_groups.nationality = 'England'::text then 1
          else null::integer
        end
      )::double precision / count(*)::double precision * 100::double precision
    )::integer
    else 0
  end as utenti_inglesi_perc,
  case
    when count(*) > 0 then (
      count(
        case
          when age_groups.nationality = 'Altra'::text then 1
          else null::integer
        end
      )::double precision / count(*)::double precision * 100::double precision
    )::integer
    else 0
  end as utenti_altra_naz_perc
from
  age_groups;
```

Comment:

This view presents the same data as the "all_users_statistics" table, but **in addition**, it shows the percentages of each data point relative to the total.

[2)answers_list_view]

code:

```
create view
  public.answers_list_view as
select
  q.id as question_id,
  q.description as question_description,
  q.color_code as question_color,
  array_agg(
    a.id
    order by
      a.lettera
  ) as answers_id,
  array_agg(
    a.description
```

```
  order by
    a.lettera
) as answers_description,
array_agg(
  a.lettera
  order by
    a.lettera
) as answers_lettera,
array_agg(
  a.voti_tot
  order by
    a.lettera
) as answers_voti_tot,
array_agg(
  a.voti_female
  order by
    a.lettera
) as answers_voti_female,
array_agg(
  a.voti_male
  order by
    a.lettera
) as answers_voti_male,
array_agg(
  a.voti_teen
  order by
    a.lettera
) as answers_voti_teen,
array_agg(
  a.voti_under30
  order by
    a.lettera
) as answers_voti_under30,
array_agg(
  a.voti_under40
  order by
    a.lettera
) as answers_voti_under40,
array_agg(
  a.voti_over40
  order by
    a.lettera
) as answers_voti_over40,
array_agg(
  a.voti_italiani
  order by
    a.lettera
) as answers_voti_italiani,
array_agg(
  a.voti_americani
```

```
  order by
    a.lettera
) as answers_voti_americani,
array_agg(
  a.voti_inglesi
  order by
    a.lettera
) as answers_voti_inglesi,
array_agg(
  a."voti_altre_nazionalità"
  order by
    a.lettera
) as "answers_voti_altre_nazionalità",
q.voti as voti_tot,
array_agg(
  case
    when q.voti > 0 then round(
      a.voti_tot::double precision / q.voti::double precision * 100::double precision
    )::integer
    else 0
  end
  order by
    a.lettera
) as answers_voti_tot_perc,
array_agg(
  case
    when sq.total_female_votes > 0::numeric then round(
      a.voti_female::double precision / sq.total_female_votes::double precision * 100::double precision
    )::integer
    else 0
  end
  order by
    a.lettera
) as answers_voti_female_perc,
array_agg(
  case
    when sq.total_male_votes > 0::numeric then round(
      a.voti_male::double precision / sq.total_male_votes::double precision * 100::double precision
    )::integer
    else 0
  end
  order by
    a.lettera
) as answers_voti_male_perc,
array_agg(
  case
    when sq.total_teen_votes > 0::numeric then round(
      a.voti_teen::double precision / sq.total_teen_votes::double precision * 100::double precision
    )::integer
    else 0
```

```
  end
  order by
    a.lettera
) as answers_voti_teen_perc,
array_agg(
  case
    when sq.total_under30_votes > 0::numeric then round(
      a.voti_under30::double precision / sq.total_under30_votes::double precision * 100::double precision
    )::integer
    else 0
  end
  order by
    a.lettera
) as answers_voti_under30_perc,
array_agg(
  case
    when sq.total_under40_votes > 0::numeric then round(
      a.voti_under40::double precision / sq.total_under40_votes::double precision * 100::double precision
    )::integer
    else 0
  end
  order by
    a.lettera
) as answers_voti_under40_perc,
array_agg(
  case
    when sq.total_over40_votes > 0::numeric then round(
      a.voti_over40::double precision / sq.total_over40_votes::double precision * 100::double precision
    )::integer
    else 0
  end
  order by
    a.lettera
) as answers_voti_over40_perc,
array_agg(
  case
    when sq.total_italian_votes > 0::numeric then round(
      a.voti_italiani::double precision / sq.total_italian_votes::double precision * 100::double precision
    )::integer
    else 0
  end
  order by
    a.lettera
) as voti_italiani_perc,
array_agg(
  case
    when sq.total_american_votes > 0::numeric then round(
      a.voti_americani::double precision / sq.total_american_votes::double precision * 100::double precision
    )::integer
    else 0
```

```
    end
    order by
      a.lettera
  ) as voti_americani_perc,
  array_agg(
    case
      when sq.total_english_votes > 0::numeric then round(
        a.voti_inglesi::double precision / sq.total_english_votes::double precision * 100::double precision
      )::integer
      else 0
    end
    order by
      a.lettera
  ) as voti_inglesi_perc,
  array_agg(
    case
      when sq.total_altre_naz_votes > 0::numeric then round(
        a."voti_altre_nazionalità"::double precision / sq.total_altre_naz_votes::double precision * 100::double
precision
      )::integer
      else 0
    end
    order by
      a.lettera
  ) as voti_altre_naz_perc,
  (
    select
      a_1.id
    from
      answers_table a_1
    where
      a_1.question_id = q.id
    order by
      a_1.voti_tot desc
    limit
      1
  ) as majority_answer_id
from
  questions_table q
  join answers_table a on q.id = a.question_id,
  (
    select
      answers_table.question_id,
      sum(answers_table.voti_female) as total_female_votes,
      sum(answers_table.voti_male) as total_male_votes,
      sum(answers_table.voti_teen) as total_teen_votes,
      sum(answers_table.voti_under30) as total_under30_votes,
      sum(answers_table.voti_under40) as total_under40_votes,
      sum(answers_table.voti_over40) as total_over40_votes,
      sum(answers_table.voti_italiani) as total_italian_votes,
      sum(answers_table.voti_americani) as total_american_votes,
```

```
    sum(answers_table.voti_inglesi) as total_english_votes,
    sum(answers_table."voti_altre_nazionalità") as total_altre_naz_votes
   from
    answers_table
   group by
    answers_table.question_id
 ) sq (
  question_id,
  total_female_votes,
  total_male_votes,
  total_teen_votes,
  total_under30_votes,
  total_under40_votes,
  total_over40_votes,
  total_italian_votes,
  total_american_votes,
  total_english_votes,
  total_altre_naz_votes
 )
where
 q.id = sq.question_id
group by
 q.id,
 q.description,
 q.color_code,
 q.voti;
```

Comment:

This view presents the response data for each question, divided into categories. Additionally, it calculates percentages, each time referring to the relevant total.

Example:

answers_description=[“**yes**”,”no”]
answers_lettera=[“**A**”,”B”]
answers_voti_tot=[**10**,20]
answers_voti_tot_perc=[**33**,67]
answers_voti_male=[**10**,10]
answers_voti_male_perc=[**50**,50]

NOTE: ***All arrays in this view must be ordered consistently***. In the previous example, the answer "yes" is letter A and has received 10 total votes.

[3)colors_list_view]

code:

```
create view
  public.colors_list_view as
select
  colors_table.id,
  colors_table.color_text,
  colors_table.color_code as "#",
  colors_table.voti_tot,
  floor(
    colors_table.voti_tot::numeric * 100.0 / (
      (
        select
          sum(colors_table_1.voti_tot) as sum
        from
          colors_table colors_table_1
      )
    )
  ) as voti_percentuali
from
  colors_table;
```

Comment:

This view presents the data from the colors_table **with the addition of percentages.**

[4)question_list_view]

code:

```
create view
  public.question_list_view as
select
  q.id,
  q.class,
  qt.topic_one,
  q.voti,
  q.color_code as color,
  q.description
from
  questions_table q
  join question_topics qt on qt.question_id = q.id;
```

Comment:

This view presents the same data as the questions_table with the addition of topic_one from the "question_topics" table.

[5)users_details_list_view]

code:

```
create view
  public.users_details_list_view as
select
  users_details.id,
  users_details.user_id,
  users_details.name,
  users_details.nickname,
  users_details.prefered_color,
  colors_table.color_code as prefered_color_code,
  users_details.gender,
  users_details.born_date,
  users_details.nationality,
 users_details.status
from
  users_details
  join colors_table on colors_table.color_text = users_details.prefered_color;
```

Comment:

This view presents the same data as the users_details table with the **addition of** prefered_color_code.

Does it make sense to have this view, or should the prefered_color_code column be added directly to the users_details table?

[6)users_friends_view]

code:

```
create view
  public.user_friends_view as
select
  friend_data.user_id,
  array_agg(
    friend_data.friend_id
    order by
      friend_data.friend_id
  ) as friends_id,
  array_agg(
```

```
    friend_data.nickname
    order by
      friend_data.friend_id
  ) as friends_nicknames,
  array_agg(
    friend_data.prefered_color
    order by
      friend_data.friend_id
  ) as prefered_colors,
  array_agg(
    colors_table.color_code
    order by
      friend_data.friend_id
  ) as prefered_colors_code
from
  (
    select
      users_friends.user1_id as user_id,
      users_friends.user2_id as friend_id,
      users_details.nickname,
      users_details.prefered_color
    from
      users_friends
      join users_details on users_friends.user2_id = users_details.user_id
    union all
    select
      users_friends.user2_id as user_id,
      users_friends.user1_id as friend_id,
      users_details.nickname,
      users_details.prefered_color
    from
      users_friends
      join users_details on users_friends.user1_id = users_details.user_id
  ) friend_data
  join colors_table on friend_data.prefered_color = colors_table.color_text
group by
  friend_data.user_id;
```

Comment:

This view presents for **each user** a list of their friends and their favorite colors.
The lists are ordered consistently!

Having the favorite colors in this view saves me from having to make 2 or more queries from Flutterflow.

3)I would like to add a column called “common_questions” that is an array ordered consistently with the other lists in the view. The value represents the number of questions we both answered.

4)I would like a column called "affinity_list" that is an array ordered consistently with the other lists in the view. The value represents a percentage of how many times two users have given the same answer to the same question.

[7)users_notifications_view]

code:

```
create view
  public.users_notifications_view as
select
  users_notifications.send_by as send_by_id,
  users_notifications.send_to as send_to_id,
  users_notifications.notification_type,
  u1.nickname as send_by_nickname,
  u2.nickname as send_to_nickname
from
  users_notifications
  join users_details u1 on users_notifications.send_by = u1.user_id
  join users_details u2 on users_notifications.send_to = u2.user_id;
```

Comment:

This view presents the same data as the "users_notifications" table with the addition of user nicknames.
Having this view saves me from having to make multiple queries on Flutterflow.

[8)users_statistics_view]

I'm unable to have all the information I want in a single table.
To create this view, I had to first create an incomplete one.
This view takes some values from the incomplete one.

Code users_statistics_view_incomplete :

```
create view
  public.users_statistics_view_incomplete as
select
  users_answers.user_id,
  users_details.nickname,
  (
    count(
      case
        when users_answers.my_answer_id = answers_list_view.majority_answer_id then 1
```

```
      else null::integer
    end
  )::numeric * 100.0 / count(*)::numeric
)::integer as majority_alignment,
(
  count(
    case
      when users_answers.mass_answer_id = answers_list_view.majority_answer_id then 1
      else null::integer
    end
  )::numeric * 100.0 / count(*)::numeric
)::integer as accuracy_rate,
(
  count(users_answers.question_id)::numeric * 100.0 / (
    (
      select
        count(*) as count
      from
        questions_table
    )
  )::numeric
)::integer as questions_completed_perc,
count(users_answers.question_id) as questions_done,
colors_table.color_code as preferred_color_code
from
users_answers
join answers_list_view on users_answers.question_id = answers_list_view.question_id
join users_details on users_answers.user_id = users_details.user_id
join colors_table on colors_table.color_text = users_details.prefered_color
join question_topics on question_topics.question_id = users_answers.question_id
group by
users_answers.user_id,
users_details.nickname,
colors_table.color_code;
```

Code users_statistics_view:

```
create view
public.users_statistics_view as
select
t.user_id,
u.nickname,
u.preferred_color_code,
array_agg(t.topic_one) as prefered_topic_one,
array_agg(t.frequency) as prefered_topic_frequency,
u.majority_alignment,
u.accuracy_rate,
u.questions_completed_perc,
u.questions_done
from
(
  select
```

```
    ua.user_id,
    qt.topic_one,
    count(*) as frequency,
    row_number() over (
      partition by
        ua.user_id
      order by
        (count(*)) desc
    ) as rank
  from
    users_answers ua
    join question_topics qt on ua.question_id = qt.question_id
  group by
    ua.user_id,
    qt.topic_one
 ) t
 join users_statistics_view_incomplete u on u.user_id = t.user_id
where
 t.rank = 1
 or t.rank = 2
 or t.rank = 3
 or t.rank = 4
 or t.rank = 5
 or t.rank = 6
 or t.rank = 7
 or t.rank = 8
 or t.rank = 9
group by
 t.user_id,
 u.nickname,
 u.majority_alignment,
 u.accuracy_rate,
 u.questions_completed_perc,
 u.preferred_color_code,
 u.questions_done;
```

Comment:

To summarize, **I want a single view with all of this data (5)** without having to use a "support" view.

NOTE: **The "prefered_topic_one" and "prefered_topic_frequency" columns must be ordered in the same way** (**from most frequent to least frequent**).

6) I also want another column called "prefered_topic_perc".
In this column, I want an array, always ordered consistently with the other 2, where instead of the number of questions answered per topic, I have the percentage of questions answered for that topic.

Example:
Attuality questions answered: 10
Politics questions answered: 6
total Attuality questions in the platform: 20
total Politics questions in the platform: 10

prefered_topic_one= [“**Attuality**”,”Politics”]
prefered_topic_frequency= [**10**,6]
prefered_topic_perc=[**50**, 60]

[9)users_friends_answers_view]

Even in this case, I'm unable to have all the informationin one view.
I want in a single table with all the data.(7)
To create this view, I had to first create an incomplete one.
This view takes some values from the incomplete one.

Code users_friends_answers_incompleted_view:

```
create view
  public.user_friends_answers_incompleted_view as
select
  friend_data.user_id,
  friend_data.question_id,
  array_agg(
    friend_data.friend_id
    order by
      friend_data.friend_id
  ) as friends_id,
  array_agg(
    friend_data.nickname
    order by
      friend_data.friend_id
  ) as friends_nicknames,
  (
    select
      array_agg(
        answers_table.lettera
        order by
          answers_table.lettera
      ) as answer_letters_sorted
    from
      answers_table
    where
      answers_table.question_id = friend_data.question_id
  ) as answer_letters,
  (
    select
```

```
    array_agg(
      answers_table.id
      order by
        answers_table.lettera
    ) as answer_ids_sorted
  from
    answers_table
  where
    answers_table.question_id = friend_data.question_id
) as answer_ids,
array_agg(
  (
    select
      users_answers_friends.user1_answer_id
    from
      users_answers_friends
    where
      users_answers_friends.user2_id = friend_data.user_id
      and users_answers_friends.question_id = friend_data.question_id
      and users_answers_friends.user1_id = friend_data.friend_id
  )
  order by
    friend_data.friend_id
) as friends_answers
from
(
  select
    users_friends.user1_id as user_id,
    users_questions_relations.question_id,
    users_friends.user2_id as friend_id,
    users_details.nickname
  from
    users_friends
    join users_details on users_friends.user2_id = users_details.user_id
    join users_questions_relations on users_friends.user1_id = users_questions_relations.user_id
    and users_questions_relations.to_public = true
  union all
  select
    users_friends.user2_id as user_id,
    users_questions_relations.question_id,
    users_friends.user1_id as friend_id,
    users_details.nickname
  from
    users_friends
    join users_details on users_friends.user1_id = users_details.user_id
    join users_questions_relations on users_friends.user2_id = users_questions_relations.user_id
    and users_questions_relations.to_public = true
) friend_data
group by
  friend_data.user_id,
  friend_data.question_id
```

```
order by
  friend_data.user_id,
  friend_data.question_id;
```

Code users_friends_answers_view:

```
create view
  public.user_friends_answers_view as
select
  uf.user_id,
  uf.question_id,
  uf.friends_id,
  uf.friends_nicknames,
  uf.answer_letters,
  uf.answer_ids,
  uf.friends_answers,
  (
    select
      count(*) as count
    from
      unnest(uf.friends_answers) answer (answer)
    where
      answer.answer is not null
  ) as voti_tot
from
  user_friends_answers_incompleted_view uf
group by
  uf.user_id,
  uf.question_id,
  uf.friends_id,
  uf.friends_nicknames,
  uf.answer_letters,
  uf.answer_ids,
  uf.friends_answers;
```

Comment:

This view contains the questions (question_id) that each user has published with their friends (users_questions_relations.to_public=True).
A) The list of the user's friends (friends_id and friends_nickname) ordered consistently.
B) The list of the answer IDs given by each friend (friends_answers) (NULL if the answer has not been given) (Ordered consistently with the previous lists)
C) The total votes received from friends (the number of answer IDs, excluding NULL).

D) The list of answer letters for the specific question (From A to Z)
E) The list of answer IDs for the specific question (Ordered consistently with the letters)

8) I want a column called "answers_voti", which is ordered consistently with D and E, which contains the number of times that answer (answers_id) appears in the friends_answers list.

9) I want another column called "answers_voti_perc", which is ordered consistently with D and E, which contains the percentage values relative to the total number of votes (C).

TRIGGERS

RispostaAggiunta table: users_answers/ event: after insert / function: RispostaAggiunta

RispostaAmicoAggiunta table: users_answers_friends/ event: after insert / function: RispostaAmicoAggiunta

RispostaCambiata table: users_answers/ event: after update / function: RispostaCambiata

UserDetailAggiunto table: users_details/ event: after insert / function: UserDetailAggiunto

FUNCTIONS

RispostaAggiunta:

code:

```
BEGIN
  -- Update total votes
  UPDATE questions_table
  SET voti = voti + 1
  WHERE questions_table.id = new.question_id;

  UPDATE answers_table
  SET voti_tot = voti_tot + 1
  WHERE answers_table.id = new.my_answer_id;

  -- Get user details
  DECLARE genere TEXT;
  DECLARE nazionalità TEXT;
  DECLARE data_nascita DATE;
  DECLARE age_years INTEGER;
  BEGIN
    SELECT gender, born_date, nationality
    INTO genere, data_nascita, nazionalità
    FROM users_details
    WHERE users_details.user_id = new.user_id;
```

```
-- Calculate age
age_years := EXTRACT(Year FROM Age(current_date, data_nascita));

-- Update votes based on gender and age
IF genere = 'Femmina' THEN
  UPDATE answers_table
  SET voti_female = voti_female + 1
  WHERE answers_table.id = new.my_answer_id;
ELSIF genere = 'Maschio' THEN
  UPDATE answers_table
  SET voti_male = voti_male + 1
  WHERE answers_table.id = new.my_answer_id;
END IF;

IF nazionalità = 'Italia' THEN
  UPDATE answers_table
  SET voti_italiani = voti_italiani + 1
  WHERE answers_table.id = new.my_answer_id;
ELSIF nazionalità = 'Usa' THEN
  UPDATE answers_table
  SET voti_americani = voti_americani + 1
  WHERE answers_table.id = new.my_answer_id;
ELSIF nazionalità = 'England' THEN
  UPDATE answers_table
  SET voti_inglesi = voti_inglesi + 1
  WHERE answers_table.id = new.my_answer_id;
ELSIF nazionalità = 'Altra' THEN
  UPDATE answers_table
  SET voti_altra_nazionalità = voti_altra_nazionalità + 1
  WHERE answers_table.id = new.my_answer_id;
END IF;

IF age_years < 20 THEN
  UPDATE answers_table
  SET voti_teen = voti_teen + 1
  WHERE answers_table.id = new.my_answer_id;
ELSIF age_years >= 30 AND age_years < 40 THEN
  UPDATE answers_table
  SET voti_under40 = voti_under40 + 1
  WHERE answers_table.id = new.my_answer_id;
ELSIF age_years >= 20 AND age_years < 30 THEN
  UPDATE answers_table
  SET voti_under30 = voti_under30 + 1
  WHERE answers_table.id = new.my_answer_id;
ELSE
  UPDATE answers_table
  SET voti_over40 = voti_over40 + 1
  WHERE answers_table.id = new.my_answer_id;
END IF;
```

```
  -- Insert into users_questions_relations
  INSERT INTO public.users_questions_relations (user_id, question_id, done, prefered, to_public)
  VALUES (new.user_id, new.question_id, true, false, false);

END;
RETURN new;
END;
```

RispostaAmicoAggiunta :

code:

```
BEGIN
  UPDATE users_answers_friends
  SET user2_answer_id = ua.my_answer_id
  FROM users_answers ua
  WHERE ua.user_id = NEW.user2_id
  AND ua.question_id = NEW.question_id
  AND users_answers_friends.question_id = NEW.question_id;

  RETURN NEW;
END;
```

RispostaCambiata:

code:

```
BEGIN
  -- Update total votes

  UPDATE answers_table
  SET voti_tot = voti_tot + 1
  WHERE answers_table.id = new.my_answer_id;

  UPDATE answers_table
  SET voti_tot = voti_tot - 1
  WHERE answers_table.id = old.my_answer_id;

  -- Get user details
  DECLARE genere TEXT;
  DECLARE nazionalità TEXT;
  DECLARE data_nascita DATE;
  DECLARE age_years INTEGER;
  BEGIN
    SELECT gender, born_date, nationality
    INTO genere, data_nascita, nazionalità
```

```
  FROM users_details
  WHERE users_details.user_id = new.user_id;

-- Calculate age
age_years := EXTRACT(Year FROM Age(current_date, data_nascita));

-- Update votes based on gender and age
IF genere = 'Femmina' THEN
  UPDATE answers_table
  SET voti_female = voti_female + 1
  WHERE answers_table.id = new.my_answer_id;
ELSIF genere = 'Maschio' THEN
  UPDATE answers_table
  SET voti_male = voti_male + 1
  WHERE answers_table.id = new.my_answer_id;
END IF;

IF genere = 'Femmina' THEN
  UPDATE answers_table
  SET voti_female = voti_female - 1
  WHERE answers_table.id = old.my_answer_id;
ELSIF genere = 'Maschio' THEN
  UPDATE answers_table
  SET voti_male = voti_male - 1
  WHERE answers_table.id = old.my_answer_id;
END IF;

IF nazionalità = 'Italia' THEN
  UPDATE answers_table
  SET voti_italiani = voti_italiani + 1
  WHERE answers_table.id = new.my_answer_id;
ELSIF nazionalità = 'Usa' THEN
  UPDATE answers_table
  SET voti_americani = voti_americani + 1
  WHERE answers_table.id = new.my_answer_id;
ELSIF nazionalità = 'England' THEN
  UPDATE answers_table
  SET voti_inglesi = voti_inglesi + 1
  WHERE answers_table.id = new.my_answer_id;
ELSIF nazionalità = 'Altra' THEN
  UPDATE answers_table
  SET voti_altra_nazionalità = voti_altra_nazionalità + 1
  WHERE answers_table.id = new.my_answer_id;
END IF;

IF nazionalità = 'Italia' THEN
  UPDATE answers_table
  SET voti_italiani = voti_italiani - 1
  WHERE answers_table.id = old.my_answer_id;
ELSIF nazionalità = 'Usa' THEN
```

```
  UPDATE answers_table
  SET voti_americani = voti_americani - 1
  WHERE answers_table.id = old.my_answer_id;
ELSIF nazionalità = 'England' THEN
  UPDATE answers_table
  SET voti_inglesi = voti_inglesi - 1
  WHERE answers_table.id = old.my_answer_id;
ELSIF nazionalità = 'Altra' THEN
  UPDATE answers_table
  SET voti_altra_nazionalità = voti_altra_nazionalità - 1
  WHERE answers_table.id = old.my_answer_id;
END IF;

IF age_years < 20 THEN
  UPDATE answers_table
  SET voti_teen = voti_teen + 1
  WHERE answers_table.id = new.my_answer_id;
ELSIF age_years >= 30 AND age_years < 40 THEN
  UPDATE answers_table
  SET voti_under40 = voti_under40 + 1
  WHERE answers_table.id = new.my_answer_id;
ELSIF age_years >= 20 AND age_years < 30 THEN
  UPDATE answers_table
  SET voti_under30 = voti_under30 + 1
  WHERE answers_table.id = new.my_answer_id;
ELSE
  UPDATE answers_table
  SET voti_over40 = voti_over40 + 1
  WHERE answers_table.id = new.my_answer_id;
END IF;

 IF age_years < 20 THEN
  UPDATE answers_table
  SET voti_teen = voti_teen - 1
  WHERE answers_table.id = old.my_answer_id;
ELSIF age_years >= 30 AND age_years < 40 THEN
  UPDATE answers_table
  SET voti_under40 = voti_under40 - 1
  WHERE answers_table.id = old.my_answer_id;
ELSIF age_years >= 20 AND age_years < 30 THEN
  UPDATE answers_table
  SET voti_under30 = voti_under30 - 1
  WHERE answers_table.id = old.my_answer_id;
ELSE
  UPDATE answers_table
  SET voti_over40 = voti_over40 - 1
  WHERE answers_table.id = old.my_answer_id;
END IF;

-- Insert into users_questions_relations
```

```
END;
RETURN new;
END;
```

UserDetailAggiunto:

code:

```
BEGIN
  -- Update colors table (incrementing voti_tot)
  UPDATE colors_table
  SET voti_tot = voti_tot + 1
  WHERE colors_table.color_text = NEW.prefered_color;

  -- Update users_statistics table (incrementing n_utenti and n_utenti_f/m)
  UPDATE all_users_statistics
  SET n_utenti = n_utenti + 1
  where all_users_statistics.id=1;

  IF NEW.gender = 'Femmina' THEN
    UPDATE all_users_statistics
    SET n_utenti_f = n_utenti_f + 1
    where all_users_statistics.id=1;
  ELSIF NEW.gender = 'Maschio' THEN
    UPDATE all_users_statistics
    SET n_utenti_m = n_utenti_m + 1
    where all_users_statistics.id=1;
  END IF;

  IF NEW.nationality = 'Italia' THEN
    UPDATE all_users_statistics
    SET n_utenti_italiani = n_utenti_italiani + 1
    where all_users_statistics.id=1;
  ELSIF NEW.nationality = 'England' THEN
    UPDATE all_users_statistics
    SET n_utenti_inglesi = n_utenti_inglesi + 1
    where all_users_statistics.id=1;
  ELSIF NEW.nationality = 'Usa' THEN
    UPDATE all_users_statistics
    SET n_utenti_americani = n_utenti_americani + 1
    where all_users_statistics.id=1;
  ELSIF NEW.nationality = 'Altra' THEN
    UPDATE all_users_statistics
    SET n_utenti_altra_nazionalità = n_utenti_altra_nazionalità + 1
    where all_users_statistics.id=1;
  END IF;

  DECLARE born_date DATE := new.born_date;  -- Remove semicolon
```

```
 DECLARE age_years INTEGER := EXTRACT(YEAR FROM Age(current_date, born_date));
 BEGIN
   IF age_years < 20 THEN
     UPDATE all_users_statistics
     SET n_utenti_teen = n_utenti_teen + 1
     where all_users_statistics.id=1;
   ELSIF age_years < 30 THEN
     UPDATE all_users_statistics
     SET n_utenti_under30 = n_utenti_under30 + 1
     where all_users_statistics.id=1;
   ELSIF age_years < 40 THEN
     UPDATE all_users_statistics
     SET n_utenti_under40 = n_utenti_under40 + 1
     where all_users_statistics.id=1;
   ELSE
     UPDATE all_users_statistics
     SET n_utenti_over40 = n_utenti_over40 + 1
     where all_users_statistics.id=1;
   END IF;
 END;


 RETURN NEW;
END;
```