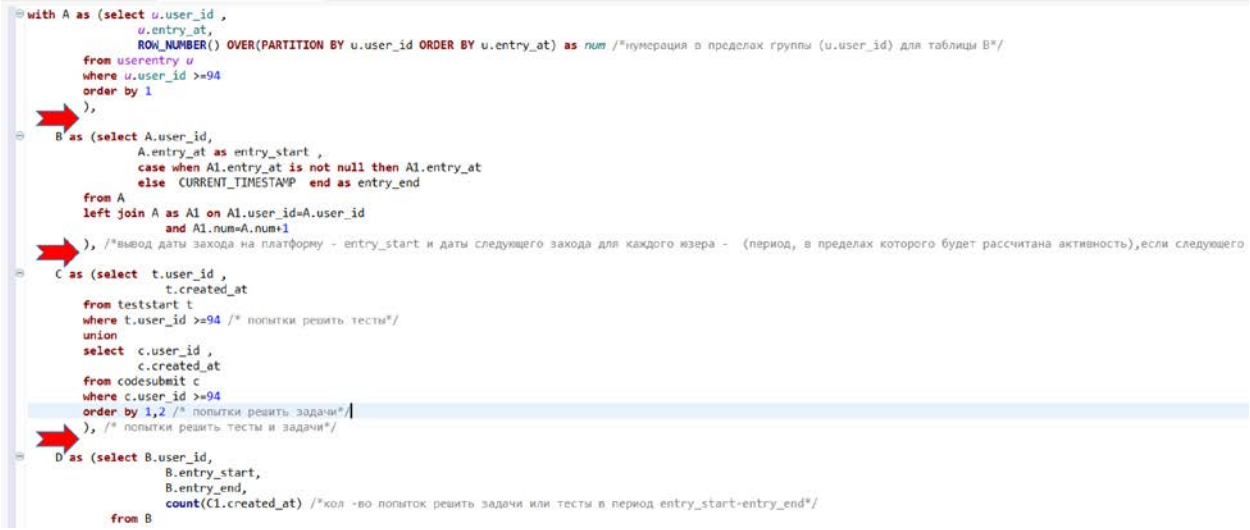
Задача 3

Как связана дата захода на платформу и активность пользователя. Под активностью имеется в виду попытка решить задачу/тест. Надо посмотреть - какой % заходов не сопровождается активностью (считаем заходы в разрезе дня: например, если в один день заходил и решал задачи, считаем, что заход сопровождался активностью).

В расчетах исключайте всегда пользователей с id < 94 - это наши внутренние аккаунты.

1. В коде присутствуют пустые строки, это не позволяет на первом шаге проверить код приходится их удалять в ручном режиме. Будьте внимательнее в будущем, чтобы не тратить время друг друга



```
with A as (select u.user_id ,
                  u.entry_at,
                  ROW_NUMBER() OVER(PARTITION BY u.user_id ORDER BY u.entry_at) as num /*нумерация в пределах группы (u.user_id) для таблицы B*/
           from userentry u
           where u.user_id >=94
           order by 1
           ),
     B as (select A.user_id,
                  A.entry_at as entry_start ,
                  case when A1.entry_at is not null then A1.entry_at
                  else  CURRENT_TIMESTAMP  end as entry_end
           from A
           left join A as A1 on A1.user_id=A.user_id
                        and A1.num=A.num+1
     ), /*вывод даты захода на платформу - entry_start и даты следующего захода для каждого юзера -  (период, в пределах которого будет рассчитана активность),если следующего
     C as (select  t.user_id ,
                   t.created_at
           from teststart t
           where t.user_id >=94 /* попытки решить тесты*/
           union
           select  c.user_id ,
                   c.created_at
           from codesubmit c
           where c.user_id >=94
           order by 1,2 /* попытки решить задачи*/
     ), /* попытки решить тесты и задачи*/
     D as (select B.user_id,
                  B.entry_start,
                  B.entry_end,
                  count(C1.created_at) /*кол -во попыток решить задачи или тесты в период entry_start-entry_end*/
           from B
```

2. Разберем отдельные моменты, пока не влияющие на функциональность кода

```
select A.user_id,
                  A.entry_at as entry_start ,
                  case when A1.entry_at is not null then A1.entry_at
                  else  CURRENT_TIMESTAMP  end as entry_end
```

Вот такую конструкцию удобнее и эстетичнее реализовать с помощью `coalesce(A1.entry_at,CURRENT_TIMESTAMP) as entry_end_too`
Согласитесь при том же результате, выглядит это приятнее и писать меньше

	user_id	entry_start	entry_end	entry_end_too
1	94	2021-11-20 10:12:36.962	2025-12-08 00:57:56.102 +0300	2025-12-08 00:57:56.102 +0300
2	95	2021-11-20 10:15:12.261	2025-12-08 00:57:56.102 +0300	2025-12-08 00:57:56.102 +0300
3	96	2021-11-20 10:22:00.110	2025-12-08 00:57:56.102 +0300	2025-12-08 00:57:56.102 +0300
4	97	2021-11-20 10:25:57.321	2021-11-22 22:04:15.835 +0300	2021-11-22 22:04:15.835 +0300
5	97	2021-11-22 22:04:15.835	2021-11-23 12:28:34.905 +0300	2021-11-23 12:28:34.905 +0300
6	97	2021-11-23 12:28:34.905	2021-11-24 11:08:47.018 +0300	2021-11-24 11:08:47.018 +0300
7	97	2021-11-24 11:08:47.018	2021-11-25 07:30:01.625 +0300	2021-11-25 07:30:01.625 +0300

```
select u.user_id ,
                  u.entry_at,
                  ROW_NUMBER() OVER(PARTITION BY u.user_id ORDER BY u.entry_at) as
num /*нумерация в пределах группы (u.user_id) для таблицы B*/
           from userentry u
           where u.user_id >=94
           order by 1
```

Использовать номера столбцов для сортировок или группировок или еще для чего-то не самая хорошая практика

В рамках данной задачи речь идет о проверках в рамках одного дня, поэтому необходимо сразу из `entry_at` извлекать дату без времени, это корректно позволит обработать пользователей, которые заходили на платформу несколько раз в день. Не понял зачем Вы подменяли null на текущую дату, видимо в этом есть какой-то сакральный смысл, но я бы предпочел этого не делать. Более того использование оконной функции lead (вывод следующего значения) позволит существенно сократить код и обойтись одним запросом. (пока я продолжаю Вашу логику, просто упрощаю ее)

```
select user_id ,
       cast (entry_at as date) as entry_at, -- вывод только даты входа
        lead(cast(entry_at as date)) over (partition by  user_id order by cast
      (entry_at as date)) as entry_end -- вывод следующей даты входа
from userentry
where user_id >=94
order by user_id
```

select distinct user_id , cast (entry_at as date) as entry_at, le | Введите

Таблица | Текст

	user_id	entry_at	entry_end
1	94	2021-11-20	[NULL]
2	95	2021-11-20	[NULL]
3	96	2021-11-20	[NULL]
4	97	2021-11-20	2021-11-22
5	97	2021-11-22	2021-11-23
6	97	2021-11-23	2021-11-24
7	97	2021-11-24	2021-11-25

Согласитесь, так проще и компактнее. Хотя если честно я еще не понял всей логики Вашего подхода, смотрим дальше.
Ага вот и логика ☺ в Таблице С (CTE) вы собираете данные о датах когда пользователь решал задачи или тесты, но тут Вы забыли учесть тот момент, что пользователь мог запускать задачу, но не отправлять на проверку или наоборот сразу отправлять на проверку, но при этом не запускать), поэтому логично соединить (union) все три таблицы. Вот так (сразу оберну это в CTE, с понятным названием)

```
with active_date as (
        select distinct
                user_id,  -- список дат, когда пользовтель хотябы раз проверял задачу
                CAST(created_at  AS date) as activ_date from  codesubmit
        union all
        select distinct
                user_id,  -- список дат, когда пользовтель хотябы раз проходил тест
                CAST(created_at  AS date) as activ_date from TestStart
        union all
        select distinct
                user_id,  -- список дат, когда пользовтель хотябы раз запускал задачу
                CAST(created_at  AS date) as activ_date from  coderun
)
```

distinct Позволит исключить повторы когда пользователь несколько раз решает один и тот же тест или задачу, это существенно сократит выдачу, например мой вариант хранит 6452 строки, а ваш даже без учета coderun 23 556 почти в ЧЕТЫРЕ раза больше, а это нагрузка на базу и уменьшение скорости выполнения. И, кстати, вы только на этом потеряли 3%
первый Ваш ответ 60%
с учетом описанной корректировки стало 57%
а по моим сведениям правильный ответ 53% - продолжим искать дальше.

Мне кстати до сих пор не понятно зачем Вы ранее создавали таблицу с id пользователя, датой захода и , главное **датой следующего захода.** Далее у Вас идут какие-то лютые джойны, в которых даже не хочется разбираться и «вишенкой» скалярное произведение таблиц

```
select round (D1.count*1.0/D2.count*100, 0)
from D1, D2
```

уверен именно там потерялось еще 4%

Предлагаю познакомится с моим вариантом, включая уже разобранный CTE запрос `active_date`

```
with active_date as (
        select distinct
                user_id,  -- список дат, когда пользовтель хотябы раз проверял задачу
                CAST(created_at  AS date) as activ_date from  codesubmit
        union all
        select distinct
                user_id,  -- список дат, когда пользовтель хотябы раз проходил тест
                CAST(created_at  AS date) as activ_date from TestStart
        union all
        select distinct
                user_id,  -- список дат, когда пользовтель хотябы раз запускал задачу
                CAST(created_at  AS date) as activ_date from  coderun
),
full_statistics as  ( --
        select distinct
                ue.user_id,
                CAST (ue.entry_at AS date) as enter_date,
                ad.activ_date
        from UserEntry ue
        left join active_date ad on ue.user_id = ad.user_id and CAST(ue.entry_at AS date) = ad.activ_date
-- проверка по юзеру и дате
        where ue.user_id>=94
),
day_stat as (
        select user_id , count(enter_date) as Enter, count(  activ_date ) as Activ -- статистика сколько
раз заходил просто, сколько раз заходил и проявлял активность
        from full_statistics
        where user_id>=94
        group by user_id
        order by user_id
)
select round (1-sum(activ)/sum(enter),2)*100 as activ_prc
from day_stat
```

full_statistics формирует таблицу где для каждого пользователя выводится дата его входа на платформу *enter_date* и если он в эту дату хоть что-то делал, то эта дата дублируется *activ_date* иначе вместо даты активности стоит null, а далее дело техники

- почитать сколько раз заходил просто, сколько раз заходил и проявлял активность (null в рассчет count не берется (*day_stat)*
- почитать долю от дней без активности от всех дней.

И никаких оконок, они здесь на мой взгляд лишние