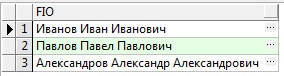
***Тест выполнил :*** *Скворцов Григорий Валерьевич*

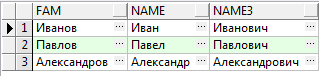
**Задание 1**

Имеется таблица table1 c полем FIO.

Содержимое данного поля имеет записи вида:



*Напишите запрос, который выводит в отдельных столбцах Фамилию, Имя и Отчество по каждому человеку*



***Ответ:***

**SELECT** F, I,

*substr*("FIO", *length*("F")+*length*("I")+2) O

**FROM**(

**SELECT** FIO, F,

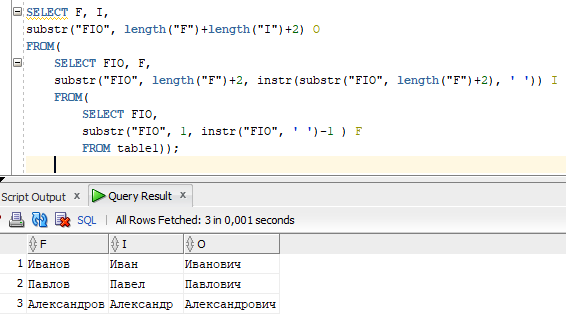
*substr*("FIO", *length*("F")+2, *instr*(*substr*("FIO", length("F")+2), ' ')) I

**FROM**(

**SELECT** FIO,

*substr*("FIO", 1, *instr*("FIO", ' ')-1 ) F

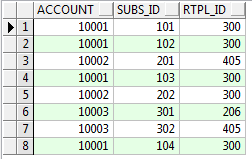
**FROM** table1));



**Задание 2**

Есть таблица вида:

create table table2 (account number, subs\_id number, rtpl\_id number),



где

subs\_id – уникальный идентификатор абонента в базе данных,

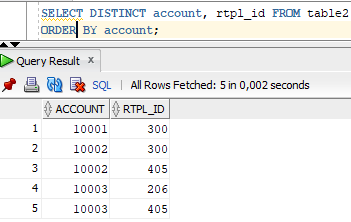
account – лицевой счет абонента, на одном лицевом счете может быть несколько абонентов,

rtpl\_id – id тарифного плана абонента.

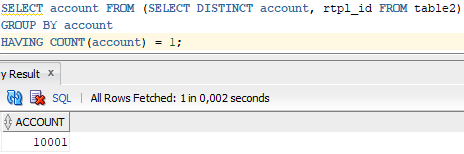
*Необходимо выбрать лицевые счета, у которых все абоненты имеют один и тот же тарифный план.*

***Ответ:***

Шаг 1. Если на лицевом счете все пользователи имеют один тарифный план, то при выборе уникальных записей по полям лицевого счета и тарифов на нём каждый лицевой счёт будет упомянут **один раз**.



Шаг 2. Теперь просто посчитаем количество упоминаний и те, что упомянуты 1 раз, то и есть ответ.



**SELECT** account **FROM** (**SELECT DISTINCT** account, rtpl\_id **FROM** table2)

**GROUP BY** account

**HAVING** **COUNT**(account) = 1;

**Задание 3**

Есть таблица вида:

create table table3 (

month\_id number,-- Месяц (например, 201501 - январь 2015 г)

type\_id number,-- Признак абонента (2 значения: 1 - старый абонент, 2 - новый абонент)

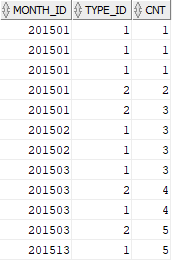
cnt number-- Кол-во абонентов

)

*Необходимо выдать результат по абонентам в следующем виде:*

*Месяц, Кол-во старых абонентов, Кол-во новых абонентов*

***Ответ:*** Рассмотрим решение на примере таблицы:



Выполним запрос:

**SELECT** olds.month\_id, olds.old, babes.baby **FROM**

(**SELECT** month\_id, count(type\_id) baby **FROM** table3

**WHERE** type\_id = 1

**GROUP BY** month\_id) babes

**INNER JOIN**

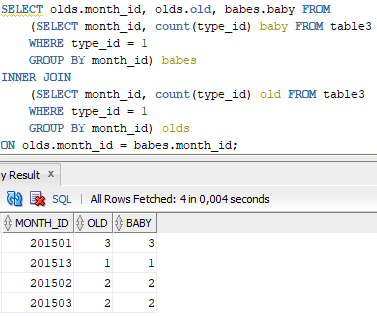
(**SELECT** month\_id, count(type\_id) old **FROM** table3

**WHERE** type\_id = 1

**GROUP BY** month\_id) olds

**ON** olds.month\_id = babes.month\_id;

Получаем результат:



**Задание 4**

Есть 2 таблицы c полями msisdn и imsi

table4

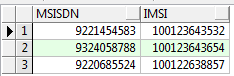
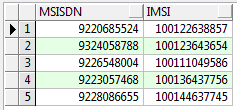


table5



*Необходимо найти записи из таблицы table1, которых нет в table 2.*

*Сколькими способами можно решить данную задачу?*

***Ответ:***

1. Можно решить через работу со множествами оператором **MINUS**
2. Можно через условие и предикат **NOT IN**
3. Можно через **LEFT OUTER JOIN** и условие **WHERE** **IS** *null*

**Задание 5**

Есть историческая таблица table6 вида:

**create table table6 (**

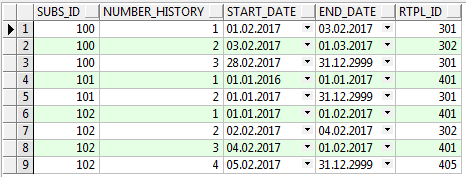
**subs\_id number,**

**number\_history number,**

**start\_date date,**

**end\_date date,**

**rtpl\_id number)**



По правилам заполнения, следующая историческая запись в этой таблице должна быть такой:

- поле start\_date следующей записи **равен** полю end\_date предыдущей записи;

- внутри одной записи соблюдается условие: start\_date **<=** end\_date.

*Необходимо найти нарушение данных условий, а именно, найти*

1. *все «дыры» (когда* ***start\_date*** *следующей записи* ***>******end\_date*** *предыдущей)*
2. *все пересечения, когда* ***временные диапазоны разных записей в истории могут пересекаться***
3. *случаи, когда не соблюдается условие* ***start\_date <= end\_date***

***Ответ:***

1. *все «дыры» (когда* ***start\_date*** *следующей записи* ***>******end\_date*** *предыдущей)*

**SELECT** thisdate.subs\_id, thisdate.number\_history,

thisdate.start\_date, thisdate.end\_date, nextdate.start\_date **AS** next\_date,

thisdate.rtpl\_id

**FROM**

(**SELECT** row\_number() **over**(**ORDER BY** table6.subs\_id)+1 **AS** num,

table6.subs\_id, table6.number\_history,

table6.start\_date, table6.end\_date, table6.rtpl\_id

**FROM** table6) thisdate

**INNER JOIN**

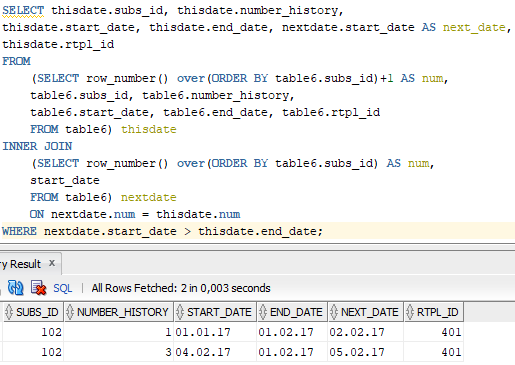
(**SELECT** row\_number() **over**(**ORDER BY** table6.subs\_id) AS num,

start\_date

**FROM** table6) nextdate

**ON** nextdate.num = thisdate.num

**WHERE** nextdate.start\_date > thisdate.end\_date;



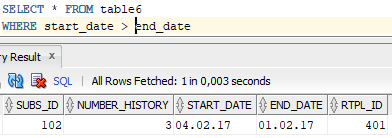
1. *все пересечения, когда* ***временные диапазоны разных записей в истории могут пересекаться***

n/a

1. *случаи, когда не соблюдается условие* ***start\_date <= end\_date***

***SELECT*** *\** ***FROM*** *table6*

***WHERE*** *start\_date > end\_date*



**Задание 6**

Есть 2 базы Oracle: db1, db2.

на db1 есть таблица table1, на db2 есть таблица table2

Требуется сделать распределенный запрос, который будет связывать две таблицы между собой.

1. *Какими правилами вы будете руководствоваться при принятии решении, если вам нужно выполнить данный запрос как можно оптимальнее?*
2. *На что будете обращать внимание и какими способами oracle можете повысить эффективность получения данных?*

***Ответ:*** n/a

**Задание 7**

*Целесообразно ли использовать индексы при обращении к таблицам, содержащим большие объемы данных? В каких случаях хорошо это или плохо? Объясните почему.*

***Ответ:***

Да, целесообразно, но есть нюансы:

* Индексы ускоряют выборки, но замедляют вставки и обновления строк, поэтому выбор каждого индекса должен быть осмыслен.
* Один из основных параметров, характеризующий индекс - **селективность** (selectivity) -**количество разных элементов в индексе**. Нет смысла индексировать поле, в котором два-три возможных значения. Пользы от такого индекса будет мало.
* Иногда имеет смысл добавить в составной индекс дополнительное поле, которое сделает индекс **покрывающим** (covering) и ускорит запросы.

**Задание 8**

*За счет чего можно добиться ускорения запросов при работе с большим объемом данных в таблице (миллионы строк). В качестве базы данных рассматривать базу Oracle.*

*Приведите несколько способов.*

***Ответ:***

Добиться ускорения запросов при работе с большим объемом данных можно следующими основными способами:

1. Правильное использование индексов
2. Правильная структура БД
3. Оптимальные SQL запросы

**1.** Правильное использование индексов

* Индексы ускоряют выборки, но замедляют вставки и обновления строк, поэтому выбор каждого индекса должен быть осмыслен.
* Один из основных параметров, характеризующий индекс - **селективность** (selectivity) -**количество разных элементов в индексе**. Нет смысла индексировать поле, в котором два-три возможных значения. Пользы от такого индекса будет мало.
* Иногда имеет смысл добавить в составной индекс дополнительное поле, которое сделает индекс **покрывающим** (covering) и ускорит запросы.

**2.** Правильная структура БД

* Стоит использовать **минимально возможные типы данных**. Чем больше тип данных, тем больше таблица, тем больше обращений к дискам нужно для получения данных.
* **Нормальные формы! Обдуманная денормализация.** Денормализовать таблицу гораздо проще, чем страдать от неоптимально денормализованной. JOIN порой работает быстрее, чем неверно денормализованные таблицы.
* Не стоит использовать NULL столбцы кроме случаев, когда они осознанно нужны.

**3.** SQL запросы

* **Циклы ресурсоёмкие**. SQL — язык множеств и к написанию запросов нужно подходить не языком функций, а языком множеств. Например, у INSERT есть синтаксис для множественной вставки. Один запрос будет выполняться на порядок быстрее, чем множество запросов в цикле
* **Стоит явно перечислять все выбираемые поля**, не злоупотреблять \*. Лишние выбранные поля замедлят запрос. Так же при добавлении новых столбцов запрос с \* может неверно работать.
* Стоит использовать **LIMIT** там, где не нужны все данные.