МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»  
(ФГБОУ ВО «ВГТУ», ВГТУ)

Факультет информационных технологий и компьютерной безопасности

Кафедра компьютерных интеллектуальных технологий проектирования

ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №9

По дисциплине: «Системы хранения и обработки данных»

Тема: «Формирование запросов к базе данных»

Выполнил работу студент группы мИИВТ-241: Боченков А.С.

подпись, дата

Принял: Короленко В.В.

подпись, дата

Воронеж 2024

Учебная задача

Формирование запросов к базе данных с учетом группировки, оконных функций, подзапросов.

1. Вывести список всех водителей, у которых количество нарушений составляет наибольшее число.

SELECT h.id, h.fio, COUNT(f.driver) AS num\_violations

FROM humans h

JOIN facts f ON h.id = f.driver

GROUP BY h.id, h.fio

HAVING COUNT(f.driver) = (SELECT MAX(num\_driver) FROM

(SELECT COUNT(\*) AS num\_driver

FROM facts f

GROUP BY driver)

);

Команда JOIN соединяет две таблицы по первичному ключу id таблицы humans с внешним ключом driver таблицы facts.

Команда GROUP BY группирует выборку по двум атрибутам id и fio таблицы humans.

Команда HAVING дает возможность задать необходимые условия по сгруппированной выборке.

SELECT COUNT(\*) AS num\_driver

FROM facts f

GROUP BY driver

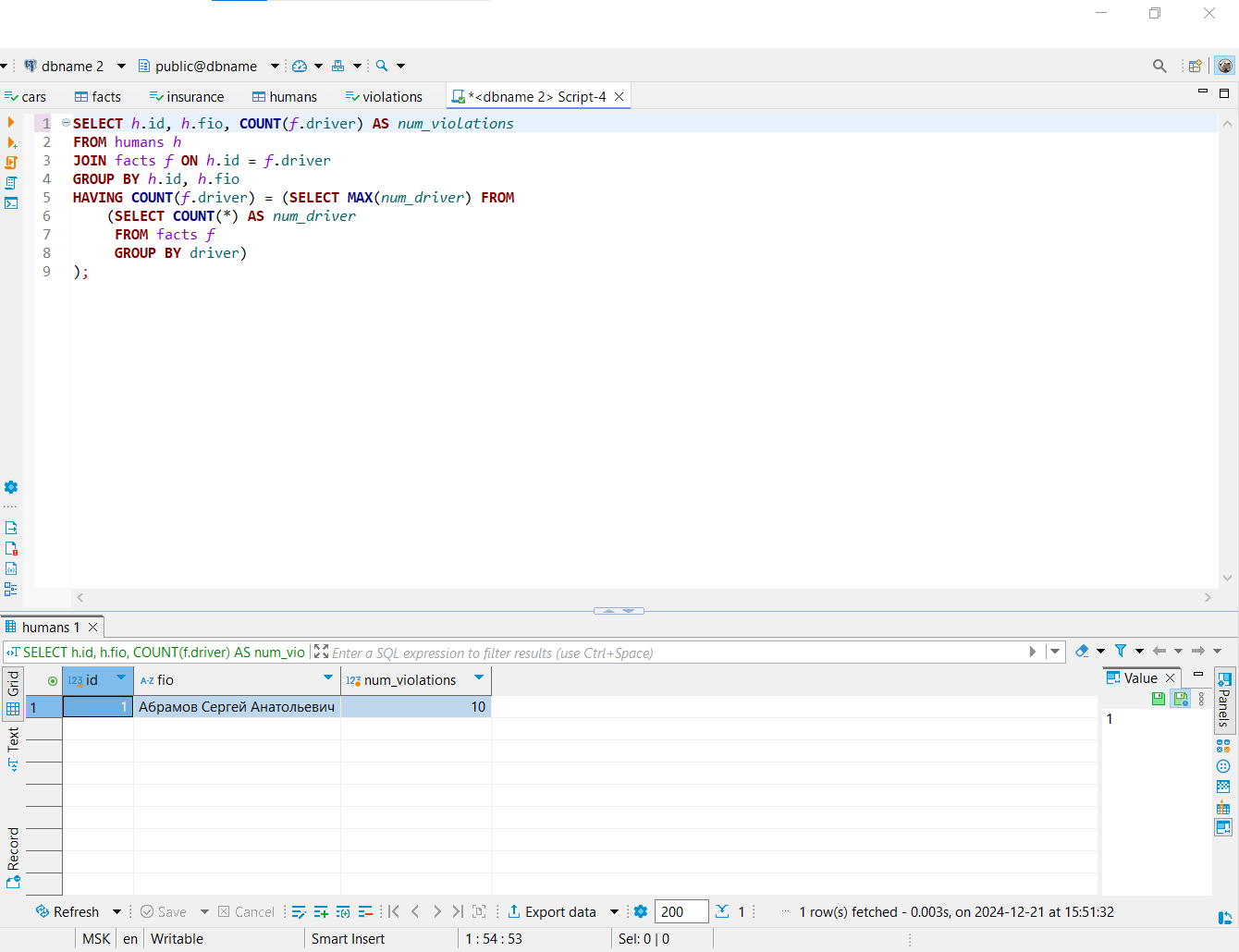
Запрос, который возвращает таблицу с общим за все годы количеством нарушений сгруппированную по каждому водителю. Здесь выступает в качестве подзапроса.

SELECT MAX(num\_driver) FROM ()

Возвращает максимальное число правонарушений из предыдущей таблицы.

COUNT(f.driver) = (SELECT MAX(num\_driver) FROM () )

Команда возьмет максимальное число нарушений и поставит в качестве условия фильтрации для исходной выборки. То есть выведет только тех водителей, с максимальным числом правонарушений.



2. Запрос, который выводит номер, ФИО и все правонарушения водителей, которые имеют максимальное число правонарушений.

SELECT h.id, h.fio, v."name", v.payment, f."date"

FROM humans h

JOIN facts f ON h.id = f.driver

JOIN violations v ON f.violation\_id = v.id

WHERE h.id IN (

SELECT h.id

FROM humans h

JOIN facts f ON h.id = f.driver

GROUP BY h.id, h.fio

HAVING COUNT(f.driver) = (SELECT MAX(num\_driver) FROM

(SELECT COUNT(\*) AS num\_driver

FROM facts f

GROUP BY driver)

)

)

ORDER BY v.payment DESC;

Как мы видим данный запрос базируется на предыдущем, поэтому объясним новые команды и общую суть.

WHERE h.id IN ()

Условие, в которое помещается список id таблицы humans, которые необходимо вывести в итоговой выборке.

ORDER BY v.payment DESC

Команда, которая выводит результат итоговой выборки в порядке убывания по атрибуту payment таблицы violations.

SELECT h.id

FROM humans h

JOIN facts f ON h.id = f.driver

GROUP BY h.id, h.fio

HAVING COUNT(f.driver) = (SELECT MAX(num\_driver) FROM

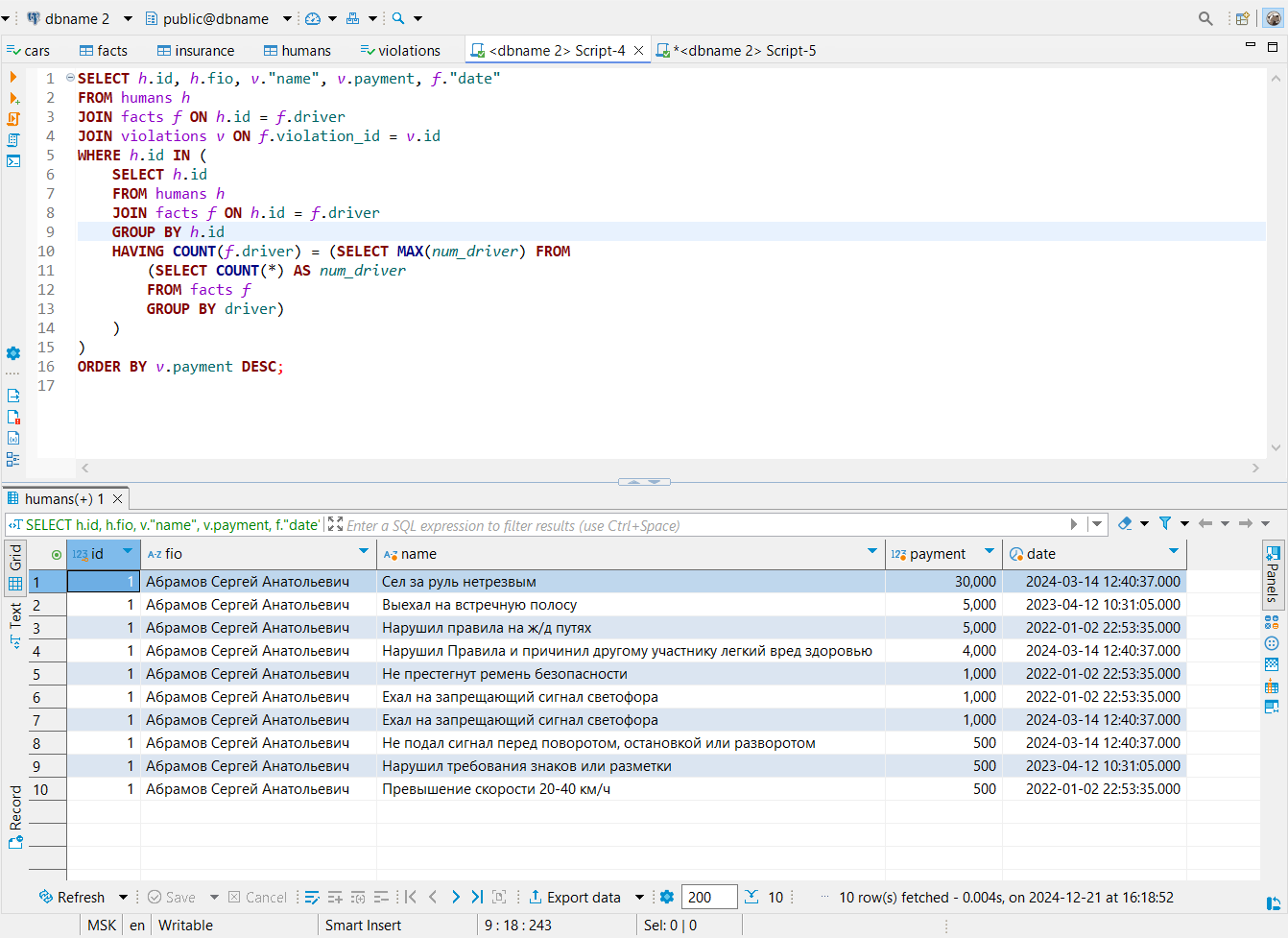
(SELECT COUNT(\*) AS num\_driver

FROM facts f

GROUP BY driver)

)

Подзапрос выводит таблицу номеров id водителей, которые имеют максимальное число правонарушений. Мы пишем запрос на вывод всех номеров водителей, потому что часто бывает ситуация, когда максимальное число имеют сразу множество объектов.



Как видим данный запрос подтверждает корректность первого, так как вывел именно все 10 правонарушений, что является максимальным числом для данного водителя и всего множества водителей в целом.

3. Запрос выводит водителей и их общую сумму штрафов, которые превышают среднюю сумму штрафов по всем водителям.

SELECT h.id, h.fio, SUM(v.payment) AS total\_payment

FROM humans h

JOIN facts f ON h.id = f.driver

JOIN violations ON f.violation\_id = v.id

GROUP BY h.id, h.fio

HAVING SUM(v.payment) > (SELECT AVG(total\_payment) FROM (

SELECT SUM(v.payment) AS total\_payment

FROM humans h

JOIN facts f ON h.id = f.driver

JOIN violations v ON f.violation\_id = v.id

GROUP BY h.id

)

)

ORDER BY total\_payment DESC;

Объясним данный запрос разделив на части, подзапросы.

SELECT SUM(v.payment) AS total\_payment

FROM humans h

JOIN facts f ON h.id = f.driver

JOIN violations v ON f.violation\_id = v.id

GROUP BY h.id

)

Данный запрос соединяет таблицы humans, facts, violations. Затем группирует выборку по людям и выводит общую сумму штрафов за все время по ним.

SELECT AVG(total\_payment) FROM ()

Из таблицы общей суммы штрафов по людям находит среднее значение суммы штрафов.

SELECT h.id, h.fio, SUM(v.payment) AS total\_payment

FROM humans h

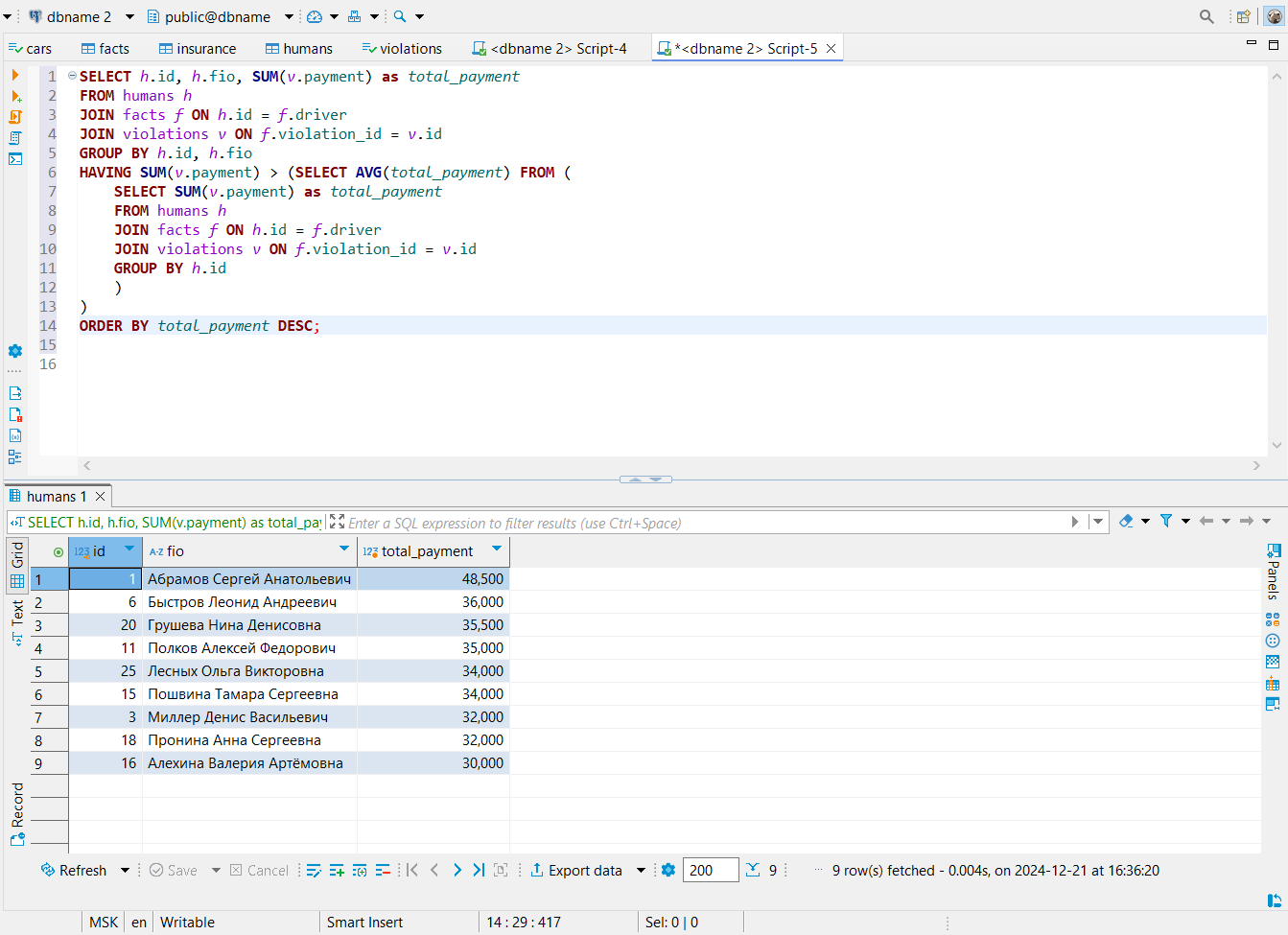
JOIN facts f ON h.id = f.driver

JOIN violations ON f.violation\_id = v.id

GROUP BY h.id, h.fio

HAVING SUM(v.payment) > (SELECT AVG(total\_payment) FROM ( ) )

В сгруппированной итоговой таблице суммы штрафов водителей вывести тех, чья сумма штрафов превышает среднюю сумму штрафов.



4. Запрос выводит общую сумму штрафов за год для каждого водителя и его скользящую сумму.

SELECT h.id, h.fio, f."date", SUM(v.payment) AS total\_payment,

SUM(SUM(v.payment)) OVER (PARTITION BY h.id ORDER BY h.id, f."date" ROWS UNBOUNDED PRECEDING) AS rolling\_age\_sum

FROM humans h

JOIN facts f ON h.id = f.driver

JOIN violations v ON f.violation\_id = v.id

GROUP BY h.id, h.fio, f."date"

ORDER BY 1, 3;

Строка, которая выводит скользящую сумму штрафов за год, выполнена через оконную функцию.

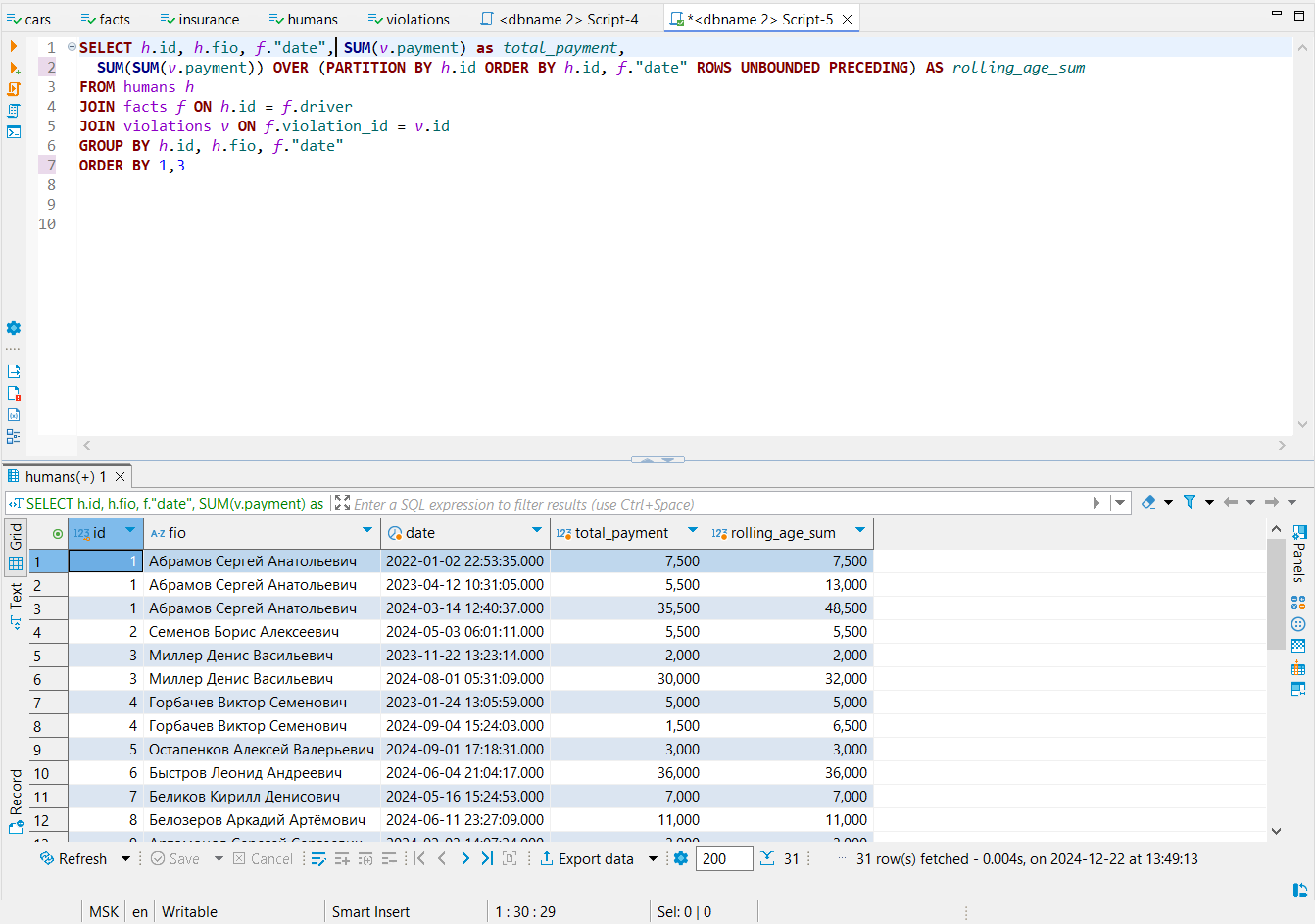
SUM(SUM(v.payment)) OVER (PARTITION BY h.id ORDER BY h.id, f."date" ROWS UNBOUNDED PRECEDING) AS rolling\_age\_sum

Команда OVER необходима для вывода оконной функции

Команда PARTITION BY нужна для группировки данных для функции агрегатной SUM(). В нашем случае сгруппировать по атрибуту id таблицы humans.

Команда ORDER BY показывает порядок вывода данных.

Команда ROWS UNBOUNDED PRECEDING говорит о том, что нужно считать функцию для номера строки каждый раз с первой для сгруппированных данных.



5. Запрос возвращает три самых дорогих штрафа для каждого водителя за все годы в порядке убывания и их скользящую сумму.

SELECT t.id, t.fio, t."date", t."name", t.payment, t.row\_payment,

SUM(SUM(t.payment)) OVER (PARTITION BY t.id ORDER BY t.id, t.row\_payment ROWS UNBOUNDED PRECEDING) AS rolling\_payment

FROM (

SELECT h.id, h.fio, f."date", v."name", v.payment,

ROW\_NUMBER() OVER (PARTITION BY h.id ORDER BY v.payment DESC) AS row\_payment

FROM humans h

JOIN facts f ON h.id = f.driver

JOIN violations v ON f.violation\_id = v.id

GROUP BY h.id, h.fio, f."date", v."name", v.payment

ORDER BY h.id, row\_payment

) t

WHERE t.row\_payment <= 3

GROUP BY t.id, t.fio, t."date", t."name", t.payment, t.row\_payment

Данный подзапрос выводит таблицу, сгруппированную по водителям и нумерует каждый штраф дополнительным номером строки.

SELECT h.id, h.fio, f."date", v."name", v.payment,

ROW\_NUMBER() OVER (PARTITION BY h.id ORDER BY v.payment DESC) AS row\_payment

FROM humans h

JOIN facts f ON h.id = f.driver

JOIN violations v ON f.violation\_id = v.id

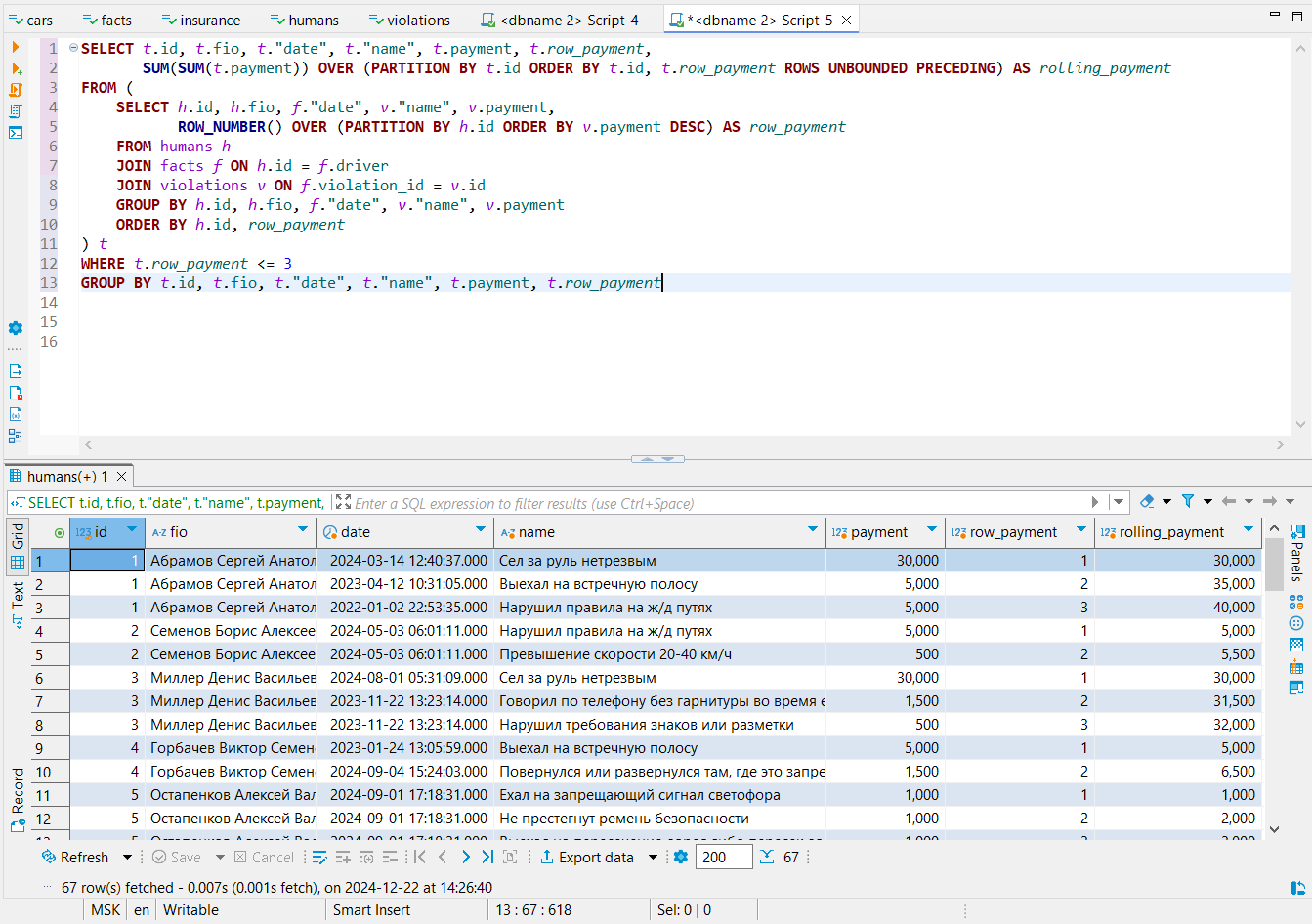
GROUP BY h.id, h.fio, f."date", v."name", v.payment

ORDER BY h.id, row\_payment

Команда ROW\_NUMBER() OVER (PARTITION BY h.id ORDER BY v.payment DESC) AS row\_payment

нумерует каждую строку дополнительно сгруппировав по номеру водителя и в порядке уменьшения штрафа.

Далее финальный запрос выводит три самых высоких штрафа со скользящей общей суммой по ним.



Контрольные вопросы

1. Что такое подзапрос?

Подзапрос – это запрос, содержащийся в другом запросе. Подзапросы окружены круглыми скобками и могут встречаться в различных частях инструкции SELECT; однако в предложении FROM подзапрос служит для создания производной таблицы, видимой для всех других предложений запроса и могущей взаимодействовать с другими таблицами, указанными в предложении FROM.

2. Как заполнить таблицу данными?

Для добавления данных в таблицу уже имеющихся атрибутов служит следующая команда INSERT INTO table\_name (agr1, agr2, … ) VALUES (new\_val1, new\_val2, … ).

3. Агрегатные функции. Основные сведения.

Агрегатные функции выполняют определенные операции над всеми строками в группе. Пример:

SELECT MAX(v.payment) AS max\_ payment,

MIN(v.payment) AS min\_ payment,

AVG(v.payment) AS avg\_ payment,

SUM(v.payment) AS tot\_ payment,

COUNT(v.payment) AS num\_ payment

FROM facts f

JOIN violations v ON f.violation\_id = v.id;

Так же используя группировку, но тогда агрегатная функция будет считать значение только по данным, входящим в группу. Пример:

SELECT h.id, h.fio, COUNT(\*) AS num\_violations

FROM humans h

JOIN facts f ON h.id = f.driver

JOIN violations v ON f.violation\_id = v.id

GROUP BY h.id, h.fio

Могут быть использованы внутри оконных функций и выборка будет сгруппирована с помощью PARTITION BY независимо от основного запроса. Пример:

SELECT h.id, h.fio, f."date", SUM(v.payment) AS total\_payment,

SUM(SUM(v.payment)) OVER (PARTITION BY h.id ORDER BY h.id, f."date" ROWS UNBOUNDED PRECEDING) AS rolling\_age\_sum

FROM humans h

JOIN facts f ON h.id = f.driver

JOIN violations v ON f.violation\_id = v.id

GROUP BY h.id, h.fio, f."date"

ORDER BY 1, 3;

4 - 5. Как объединить таблицы в запросе? Способы объединения таблиц. Краткая характеристика.

Объединить таблице в запросе можно несколькими способами с помощью JOIN, WHERE, UNION.

Результирующие таблицы после запросов с помощью JOIN, WHERE аналогичны. Пример:

SELECT MAX(v.payment) AS max\_ payment

FROM facts f

JOIN violations v ON f.violation\_id = v.id;

SELECT MAX(v.payment) AS max\_ payment

FROM facts f, violations v

WHERE f.violation\_id = v.id;

Можно объединить в одной результирующей таблице несколько запросов к разным даже не связанным таблицам, главное, чтобы количество результирующих аргументов и их тип совпадали. Следующий пример объединит в одном столбце максимальное и минимальное значение после двух разных запросов.

SELECT MAX(v.payment) AS payment

FROM facts f

JOIN violations v ON f.violation\_id = v.id

UNION

SELECT MIN(v.payment) AS payment

FROM facts f

JOIN violations v ON f.violation\_id = v.id;

6 - 7. План запроса. Оптимизация запроса. Краткое описание.

Каждый раз, когда запрос отправляется на сервер, сервер перед выполнением запроса проверяет следующее.

* Есть ли разрешение на выполнение инструкции.
* Есть ли разрешение на доступ к нужным данным.
* Правильность синтаксиса инструкции.

Если инструкция успешно проходит эти три теста, запрос передается *оптимизатору запросов*, задача которого – определить наиболее эффективный способ выполнения этого запроса. Оптимизатор изучает, например, порядок соединения таблиц, указанных в конструкции FROM и какие индексы доступны, а затем выбирает *план выполнения запроса*, который сервер осуществляет при выполнении запроса. Как только сервер завершает выполнение запроса, вызывающему приложению возвращается результирующий набор.

Для того, что узнать, как именно оптимизируется запрос указывается инструкция EXPLAIN и как расширение можно указать дополнительный параметр (ANALYSE, VERBOSE, COSTS, BUFFERS, TIMING, FORMAT). Точность оценок планировщика можно проверить, используя команду EXPLAIN с параметром ANALYZE. С этим параметром EXPLAIN на самом деле выполняет запрос, а затем выводит фактическое число строк и время выполнения, накопленное в каждом узле плана, вместе с теми же оценками, что выдаёт обычная команда EXPLAIN.

8. Оконные функции. Краткое описание.

Вызов оконной функции представляет собой применение функции, подобной агрегатной, к некоторому набору строк, выбранному запросом. В отличие от вызовов не оконных агрегатных функций, при этом не происходит группировка выбранных строк в одну — каждая строка остаётся отдельной в результате запроса. Однако оконная функция имеет доступ ко всем строкам, вошедшим в группу текущей строки согласно указанию группировки (списку PARTITION BY) в вызове оконной функции. Вызов оконной функции может иметь следующие формы:

имя\_функции ([выражение [, выражение ... ]]) [ FILTER ( WHERE предложение\_фильтра ) ] OVER имя\_окна

имя\_функции ([выражение [, выражение ... ]]) [ FILTER ( WHERE предложение\_фильтра ) ] OVER ( определение\_окна )

имя\_функции ( \* ) [ FILTER ( WHERE предложение\_фильтра ) ] OVER имя\_окна

имя\_функции ( \* ) [ FILTER ( WHERE предложение\_фильтра ) ] OVER ( определение\_окна )

имя\_окна — ссылка на именованное окно, определённое предложением WINDOW в данном запросе. Также возможно написать в скобках полное определение\_окна, используя тот же синтаксис определения именованного окна в предложении WINDOW; подробнее это описано в справке по SELECT. Стоит отметить, что запись OVER имя\_окна не полностью равнозначна OVER (имя\_окна ...); последний вариант подразумевает копирование и изменение определения окна и не будет допустимым, если определение этого окна включает определение рамки.

Указание PARTITION BY группирует строки запроса в разделы, которые затем обрабатываются оконной функцией независимо друг от друга. PARTITION BY работает подобно предложению GROUP BY на уровне запроса, за исключением того, что его аргументы всегда просто выражения, а не имена выходных столбцов или числа. Без PARTITION BY все строки, выдаваемые запросом, рассматриваются как один раздел. Указание ORDER BY определяет порядок, в котором оконная функция обрабатывает строки раздела. Оно так же подобно предложению ORDER BY на уровне запроса и так же не принимает имена выходных столбцов или числа. Без ORDER BY строки обрабатываются в неопределённом порядке.

9. Команда и синтаксис для группировки данных.

Сгруппировать данные можно с помощью команды GROUP BY ([арг таб, аргтаб]) в основном запросе. Так же в внутри оконной функции с помощью указания PARTITION BY, OVER (PARTITIOON BY [арг, арг]).

10. Сортировка данных. Краткое описание.

После того как запрос выдал таблицу результатов (после обработки списка выборки), её можно отсортировать. Если сортировка не задана, строки возвращаются в неопределённом порядке. Фактический порядок строк в этом случае будет зависеть от плана соединения и сканирования, а также от порядка данных на диске, поэтому полагаться на него нельзя. Определённый порядок выводимых строк гарантируется, только если этап сортировки задан явно.

Порядок сортировки определяет предложение ORDER BY:

ORDER BY выражение\_сортировки1 [ASC | DESC] [NULLS { FIRST | LAST }]

[, выражение\_сортировки2 [ASC | DESC] [NULLS { FIRST | LAST }] ...]

Когда указывается несколько выражений, последующие значения позволяют отсортировать строки, в которых совпали все предыдущие значения. Каждое выражение можно дополнить ключевыми словами ASC или DESC, которые выбирают сортировку соответственно по возрастанию или убыванию. По умолчанию принят порядок по возрастанию (ASC). При сортировке по возрастанию сначала идут меньшие значения, где понятие «меньше» определяется оператором <. Подобным образом, сортировка по возрастанию определяется оператором >.

Для определения места значений NULL можно использовать указания NULLS FIRST и NULLS LAST, которые помещают значения NULL соответственно до или после значений не NULL. По умолчанию значения NULL считаются больше любых других, то есть подразумевается NULLS FIRST для порядка DESC и NULLS LAST в противном случае.

11. Условия для выборки данных (WHERE, HAVING). Краткое описание.

Предложение WHERE – это механизм для фильтрации нежелательных строк из результирующего набора. С данным предложением можно использовать операторы AND, OR и указать массив значений с помощью оператора IN.

При группировке данных может потребоваться отфильтровать нежелательные данные из набора результатов на основе групп данных, а не необработанных данных. Поскольку предложение GROUP BY выполняется после того, как вычислено предложение WHERE, добавить условия фильтрации к предложению WHERE для этой цели нельзя. Вместо этого нужно поместить условия группового фильтра в предложение HAVING.