Лабораторная работа № 7 по теме: «простые SQL-запросы»

Цель: научится получать данные из БД с помощью запросов

Ход работы:

1. Оформите титульный лист
2. Выполните задания (по вариантам) и оформите отчет
3. Сдайте лабораторную работу преподавателю (на следующей паре)

Отчет по лабораторной работе должен содержать (выполняется в MS Word):

1. ФИ, группа в верхнем колонтитуле на каждой странице, кроме первой
2. Первая страница – титульный лист
3. Каждая страница пронумерована, кроме первой
4. Выполненные задания начиная со второй страницы документа (каждое задание подписано, при необходимости присутствуют скриншоты) **P.S. Скриншоты делаются ВСЕГО экрана компьютера, а не только окна программы или строки кода!!!**

Теоретическая часть:

Процесс или команда получения данных из базы данных называется запросом. В SQL запросы формулируются с помощью команды SELECT.

Простой запрос выглядит так:

```
SELECT * FROM table1;
```

Если предположить, что в базе данных есть таблица table1, эта команда получит все строки с содержимым всех столбцов из table1.

Предложение FROM образует таблицу из одной или нескольких ссылок на таблицы, разделённых запятыми.

```
FROM табличная_ссылка [, табличная_ссылка [, ...]]
```

Для обновления данных в БД используется оператор UPDATE.

UPDATE имя_таблицы
SET столбец1 = значение1, столбец2 = значение2, ... столбецN = значениеN
[**WHERE** условие_обновления]

```
UPDATE tcust SET phone = '352 3442' WHERE fname = 'Peter' and lname = 'Bradley';
```

То есть в столбце tcust установим новое значение поля phone для определенного человека.

ORDER BY, ASC и DESC

При выполнении SELECT запроса, строки по умолчанию возвращаются в неопределенном порядке. Фактический порядок строк в этом случае зависит от плана соединения и сканирования, а также от порядка расположения данных на диске, поэтому полагаться на него нельзя. Для упорядочивания записей используется конструкция ORDER BY.

SELECT * FROM ticket_flights WHERE amount < 5000 AND fare_conditions = 'Economy' ORDER BY amount DESC\ASC; (если не будет указано как упорядочить, то по умолчанию ASC)

Последовательность запроса такова:

С помощью SELECT отбирается информация, которая будет отображена в итоговой таблице. В этом запросе это вся (для этого после SELECT нужно поставить символ умножения «*») информация о билете, который прошел фильтр.

• В блоке WHERE пишется информация, по которой будет проходить выборка из таблицы. Как и писалось выше это стоимость не выше 5000 и эконом класс.

• Далее в строке после ORDER BY выбирается по какому полю будет идти сортировка в нашем случае указываем стоимость, и ставим ключевое слово DESC что будет означать сортировку по убыванию.

GROUP BY

Команда GROUP BY позволяет группировать результаты при выборке из базы данных. Группировку лучше всего использовать в месте с агрегатными функциями по типу: COUNT (подсчёт кол-ва строк), SUM (сумма значений), MAX (максимальное значение в строках), MIN (минимальное значение строках), AVG (среднее значение). Пример такого запроса может звучать так:

```
SELECT fare_conditions, COUNT(*) FROM seats GROUP BY fare_conditions;
```

Последовательность запроса такова:

• С помощью SELECT отбирается информация, которая будет отображена в итоговой таблице. В этом запросе это названия групп (обязательное поле, когда используется сгруппированных строк с помощью функции COUNT так же даем этому столбцу название «seats»

• В строке с FROM выбирается таблица для группировки

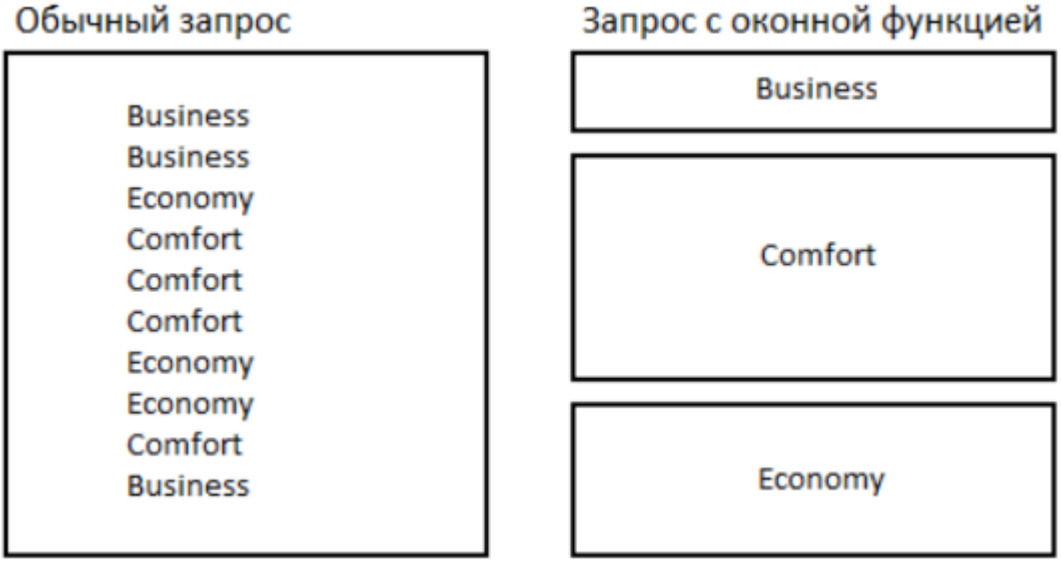
• Далее в строке после GROUP BY пишем название столбца, по которому будет проведена группировка

Функция COUNT принимает один из нескольких параметров:

- «*» – означает то, что функция COUNT возвращает все записи в таблице;
- column_name – функция COUNT возвращает количество записей конкретного столбца (только NOT NULL);
- DISTINCT column_name – функция COUNT возвращает количество только разных записей конкретного столбца (только NOT NULL).

Оконные функции (OVER)

При обычном запросе все множество строк обрабатывается как бы единым «цельным куском», для которого считаются агрегаты. А при использовании оконных функций, запрос делится на части (окна) и уже для каждой из отдельных частей считаются свои агрегаты.




```
SELECT *, max(amount) OVER(PARTITION BY fare_conditions ORDER BY amount) as Data FROM ticket_flights;
```

С помощью SELECT отбираем все столбцы таблицы и дополняем это оконной функцией, которая добавит столбец максимального значения стоимости (так как оконные функции работает только с агрегатными функциями), однако благодаря этому элементы таблицы сгруппируются по столбцу после ключевого слова PARTITION BY (fare_conditions) и от сортируются по столбцу после ORDER BY (amount). (как параметра сортировки, так и группировки может и не быть вовсе). Так же называем новый столбец с помощью AS и берем данные из таблицы с помощью FROM.

DISTINCT

DISTINCT — это предложение в PostgreSQL, которое используется для удаления дубликатов строк из набора результатов запроса.

Синтаксис:

— для одного столбца: SELECT DISTINCT column_1 FROM table_name;

— для нескольких столбцов: SELECT DISTINCT column_1, column_2, column_3 FROM table_name;

«Чтобы выбрать строки, у которых столбец не равен ни одному из предложенных значений, используйте ключевое слово **NOT IN**: **SELECT * FROM employees WHERE last_name NOT IN ('Ivanov', 'Abramov', 'Sutkin');**

Этот пример вернёт все строки из таблицы employee, где last_name не равно «Ivanov», «Abramov» или «Sutkin».

ИЛИ: **SELECT * FROM contacts WHERE address IS NOT NULL**; Этот пример вернёт все записи из таблицы contacts, где address не содержит значения NULL.

Операторы фильтрации

Оператор IN позволяет определить набор значений, которые должны иметь столбцы:

WHERE выражение [NOT] IN (выражение)

Выражение в скобках после IN определяет набор значений. Этот набор может вычисляться динамически на основании, например, еще одного запроса, либо это могут быть константные значения.

Например, выберем товары, у которых производитель либо Samsung, либо Xiaomi, либо Huawei:

SELECT * FROM Products WHERE Manufacturer IN ('Samsung', 'HTC', 'Huawei');

В качестве алтернативы можно было бы проверить все эти значения через оператор OR: **SELECT * FROM Products WHERE Manufacturer = 'Samsung' OR Manufacturer = 'HTC' OR Manufacturer = 'Huawei';**

Оператор BETWEEN определяет диапазон значений с помощью начального и конечного значения, которому должно соответствовать выражение:

WHERE выражение [NOT] BETWEEN начальное_значение AND конечное_значение

Например, получим все товары, у которых цена от 20 000 до 50 000 (начальное и конечное значения также включаются в диапазон):

SELECT * FROM Products WHERE Price BETWEEN 20000 AND 50000;

Также можно использовать более сложные выражения. Например, получим товары, запасы которых на определенную сумму (цена * количество):

SELECT * FROM Products WHERE Price * ProductCount BETWEEN 90000 AND 150000;

Оператор LIKE

Оператор LIKE принимает шаблон строки, которому должно соответствовать выражение.

WHERE выражение [NOT] LIKE шаблон_строки

Для определения шаблона могут применяться ряд специальных символов подстановки:

%: соответствует любой подстроке, которая может иметь любое количество символов, при этом подстрока может и не содержать ни одного символа

Например, выражение **WHERE ProductName LIKE 'Galaxy%'** соответствует таким значениям как "Galaxy Ace 2" или "Galaxy S7"

_: соответствует любому одиночному символу

Например, выражение **WHERE ProductName LIKE 'Galaxy S_'** соответствует таким значениям как "Galaxy S7" или "Galaxy S8".

Дата и время

В PostgreSQL есть четыре основных типа для работы с временем:

- **date** - дата без временной части. Дату удобно задавать в формате 'YYYY-MM-DD' ('2010-06-30')
- **timestamp** - дата с временем. Данный тип данных отличается от типа date наличием временной составляющей, которая позволяет хранить время с точностью до микросекунды.
- **interval** - интервал времени между датами. Обычно задается в достаточно свободном текстовом формате (примеры: interval '1 day 2 week 36 minute', interval '1 year -5 day 1 hour')
- **time** - время без даты. Удобный тип данных чем-то похожий на interval, который работает по модулю 24 (часа), то есть интервал, в котором отсутствуют дни.

Временная арифметика

Вычисление интервала доступно для всех временных типов данных.

Если в операции участвуют только переменные с типом date, то результатом будет целое число, означающее количество дней между указанными датами: **date - date = число дней (integer)**

Если при вычислении интервала между датами хотя бы одна переменная с типом timestamp, то результатом вычисления будет значение с типом interval: **timestamp - timestamp = interval**

В арифметических временных операциях можно произвольно смешивать временные типы данных. В этом случае date преобразуется в timestamp.

Значения времени можно уменьшать или увеличивать на заданный интервал, результатом будет значение с типом timestamp.

SELECT '2012-01-05'::timestamp - '1 hour'::interval AS result;

```
      result
---------------------
2012-01-04 23:00:00
```

SELECT '2010-05-06'::date + interval '1 month 1 day 1 minute' AS result;

```
       result
---------------------
2010-06-07 00:01:00
```

Интервалы можно складывать и вычитать между собой, делить и умножать на произвольные вещественные числа.

SELECT '1 hour'::interval / 7 AS result;

```
    result
-----------------
00:08:34.285714
```

SELECT interval '1 minute' * 99 AS result;

```
 result
----------
01:39:00
```

SELECT interval '1 hour' - interval '33 minutes' AS result;

```
 result
----------
00:27:00
```

Интервалы хранят данные в трёх отдельных полях: месяцах, днях, секундах. Это сделано из-за того, что количество дней в месяце и часов в дне могут быть разными. Поэтому арифметика с участием двух интервалов довольно специфичная:

SELECT interval '1 day 20 hours' + interval '10 hours' AS result;

```
   result
-------------
day 30:00:00
```

SELECT interval '20 days' + interval '20 days' AS result;

```
 result
---------
40 days
```

Специальные значения времени

today - возвращает текущую дату

```
SELECT 'today'::timestamp, now()::date;
      timestamp      |    now
---------------------+-----------
```

```
 2013-07-21 00:00:00 | 2013-07-21
```

now - возвращает текущую дату с временем

```
SELECT 'now'::timestamp, now();
         timestamp          |              now
----------------------------+-------------------------------
 2013-07-21 17:16:55.938193 | 2013-07-21 17:16:55.938193+06
```

tomorrow - возвращает дату завтрашнего дня

```
SELECT 'tomorrow'::timestamp, now()::date;
      timestamp      |    now
---------------------+-----------
 2013-07-22 00:00:00 | 2013-07-21
```

yesterday - возвращает дату вчерашнего дня

```
SELECT 'yesterday'::timestamp, now()::date;
      timestamp      |    now
---------------------+-----------
 2013-07-20 00:00:00 | 2013-07-21
```

Для округления времени существует полезная функция date_trunc

```
SELECT date_trunc('month', timestamp '2010-06-12 20:11');
     date_trunc
---------------------
 2010-06-01 00:00:00
```

```
SELECT date_trunc('week', timestamp '2010-06-12 20:11');
     date_trunc
---------------------
 2010-06-07 00:00:00
```

```
SELECT date_trunc('year', timestamp '2010-06-12 20:11');
     date_trunc
---------------------
 2010-01-01 00:00:00
```

Для получение полей времени применяются функции EXTRACT или date_part

```
SELECT EXTRACT(year FROM now()), date_part('year', now()), now()::date;
 date_part | date_part |   now
-----------+-----------+-----------
      2013 |      2013 | 2013-07-21
```

```
SELECT EXTRACT(month FROM now()), date_part('month', now()), now();
 date_part | date_part |   now
-----------+-----------+-----------
         7 |         7 | 2013-07-21
```

Агрегатные функции SQL

В SQL имеется пять агрегатных функций: COUNT, SUM, AVG, MAXи MIN. Они выполняют различные действия над результатами оператора SELECT.

Функция COUNT работает вне зависимости от типа данных столбца, а функции SUM, AVG, MAX и MIN оперируют только числовыми столбцами (integer, numeric и т. д.).

Функция COUNT подсчитывает количество строк в результате, а функция SUM вычисляет сумму значений числового столбца.

Например, следующий SQL-запрос подсчитывает количество строк в таблице: **SELECT COUNT (*) FROM product;**

Рассмотрим следующие 2 запроса:

SELECT COUNT(Name_product) FROM product; Результат: 8

SELECT COUNT (Distinct Name_product) FROM product; Результат: 5

Разница в результатах возникает потому, что второй оператор SELECT не учитывает повторяющиеся строки.

Ход работы.

ЗАДАНИЕ (каждый запрос сопроводить скриншотом)

Для заполненной БД из ЛР 6 (про галерею) придумать не менее 10 запросов, используя информацию из теории. Обязательно проделать следующие типы запросов:

1. сортировка (по возрастанию и по убыванию) и группировка с подсчетом выведенных строк
2. использование агрегатных функций
3. не менее 2х запросов с датами
4. не менее 3х запросов с операторами фильтрации и LIKE
5. один запрос на обновление (скрин до обновления и после обновления)
6. применение оконной функции
7. вывод информации без повторяющихся строк

Критерии оценки:

Зачтено	Все задания лабораторной работы самостоятельно, студент может объяснить, как он выполнил то или иное действие; отчет содержит все необходимые сведения, студент уверенно отвечает на вопросы по теории
Не зачтено	Лабораторная работа выполнена не самостоятельно (студент не может ответить ни на один вопрос преподавателя), или выполнена не до конца, или не выполнена совсем