**Лабораторные работы по курсу**

**Базы данных**

**Лабораторная работа 2**

**«Оператор SELECT языка SQL»**

**Москва, 2023**

Оглавление

[1. Теоретическая часть 3](#_Toc138942891)

[1.1. Фильтрация строк 3](#_Toc138942892)

[1.2. Условия отбора строк 4](#_Toc138942893)

[1.3. Проверка на членство во множестве 4](#_Toc138942894)

[1.4. Поиск с использованием шаблона 5](#_Toc138942895)

[1.5. Вычисляемые столбцы 6](#_Toc138942896)

[1.6. Агрегирование и группировка 7](#_Toc138942897)

[1.7. Сортировка 8](#_Toc138942898)

[1.8. Оконные функции 9](#_Toc138942899)

[1.9. Несколько важных замечаний 9](#_Toc138942900)

[2. Практическая часть 12](#_Toc138942901)

[2.1. Задание 1 12](#_Toc138942902)

[2.2. Задание 2 12](#_Toc138942903)

[2.3. Задание 3 12](#_Toc138942904)

[2.4. Задание 4 12](#_Toc138942905)

[2.5. Задание 5 12](#_Toc138942906)

[Список литературы 13](#_Toc138942907)

1. Теоретическая часть

Основой работы с базами данных является умение правильно составить запрос на фильтрацию данных. Для данной задачи используется оператор SELECT. Ниже представлен сокращенный синтаксис данного оператора. Более подробно можно прочитать в документации [1].

SELECT [ ALL | DISTINCT [ ON ( выражение [, ...] ) ] ]

[ \* | выражение [ [ AS ] имя\_результата ] [, ...] ]

[ FROM элемент\_FROM [, ...] ]

[ WHERE условие ]

[ GROUP BY элемент\_группирования [, ...] ]

[ HAVING условие ]

[ { UNION | INTERSECT | EXCEPT } [ ALL | DISTINCT ] выборка ]

[ ORDER BY выражение [ ASC | DESC | USING оператор ] ]

[ LIMIT { число | ALL } ]

[ OFFSET начало [ ROW | ROWS ] ]

Оператор SELECTполучает строки из базы данных и возвращает их в виде таблицы результатов запроса. Результат выполнения запроса может оказаться пустым множеством.

Рассмотрим выполнение SQL запроса на фильтрацию данных по шагам.

* 1. Фильтрация строк

Первыми и обязательными элементами запроса являются ключевые слова SELECT и FROM. После ключевого слова SELECT указывается список столбцов, значения которых необходимо вывести. Возвращаемые столбцы могут быть получены непосредственно из базы данных или вычислены во время выполнения запроса. Для краткого обозначения того, что необходимо вывести все столбцы таблицы используют символ \*.

После ключевого слова FROM указывается имя таблицы, из которой будет производиться выборка данных. Данная таблица может быть реальной таблицей из базы данных, или виртуальной, полученной в результате подзапроса.

Рассмотрим пример. [[1]](#footnote-1)

SELECT \* FROM student;

student\_id | surname | name | patronymic | students\_group\_number | birthday | email

------------+-----------+--------+-------------+-----------------------+------------+------------------------

838389 | Иванова | Есения | Ивановна | ИБ-21 | 2003-12-23 | IvanovaEsenija@miet.ru

822557 | Федоров | Роберт | Даниилович | ИБ-21 | 2003-12-01 | FedorovRobert@miet.ru

820564 | Морозов | Илья | Глебович | ИБ-21 | 2003-08-17 | MorozovIlja@miet.ru

830220 | Жилин | Никита | Максимович | ИБ-21 | 2003-04-03 | ZhilinNikita@miet.ru

832502 | Худяков | Андрей | Львович | ИБ-21 | 2003-01-05 | HudjakovAndrej@miet.ru

850203 | Плотников | Марк | Арсентьевич | ИБ-21 | 2003-04-11 | PlotnikovMark@miet.ru

832928 | Максимов | Виктор | Фёдорович | ИБ-21 | 2003-07-11 | MaksimovViktor@miet.ru

856305 | Баранов | Максим | Маркович | ИБ-21 | 2003-09-11 | BaranovMaksim@miet.ru

Данный запрос возвращает список всех студентов университета. Для того, чтобы вывести только ФИО студентов, необходимо вместо знака \* указать названия требуемых полей.

SELECT surname, name, patronymic

FROM student;

surname | name | patronymic

-----------+--------+-------------

Иванова | Есения | Ивановна

Федоров | Роберт | Даниилович

Морозов | Илья | Глебович

Жилин | Никита | Максимович

Худяков | Андрей | Львович

Плотников | Марк | Арсентьевич

Максимов | Виктор | Фёдорович

Баранов | Максим | Маркович

Следующий запрос

SELECT name FROM student;

Выведет все имена студентов, с учетом повторений. Чтобы их исключить возможно использовать ключевое слово DISTINCT

SELECT DISTINCT name FROM student;

* 1. Условия отбора строк

Для наложения условий поиска используется оператор WHERE. В качестве операторов сравнения используются операторы: =, <>,>,> =, <, <=.

Например, чтобы найти всех студентов с именем «Сергей», возможно выполнить следующий запрос:

SELECT surname, name, patronymic

FROM student

WHERE name = 'Сергей';

surname | name | patronymic

----------+--------+--------------

Дорофеев | Сергей | Евгеньевич

Соловьев | Сергей | Владимирович

(2 rows)

Обратите внимание, что сравниваемая строка заключается в одинарные кавычки. Об их использовании подробнее будет рассказано ниже.

Аналогично можно найти всех студентов, у которых номер студенческого билета больше числа 850000.

SELECT surname, name, patronymic

FROM student

WHERE student\_id > 850000;

surname | name | patronymic

-----------+---------+-------------

Плотников | Марк | Арсентьевич

Баранов | Максим | Маркович

Носов | Иван | Глебович

Николаев | Андрей | Матвеевич

Николаев | Алексей | Серафимович

Для наложения нескольких условий возможно воспользоваться логическими операторами – AND, OR, NOT.

Например, следующий запрос вернет ФИО студентов, чей день рождения попадает в диапазон с 25 июня 2002 до 25 июня 2003 года включительно.

SELECT surname, name, patronymic

FROM student

WHERE birthday > '25/06/2002' AND birthday < '25/06/2003';

surname | name | patronymic

-----------+--------+-------------

Жилин | Никита | Максимович

Худяков | Андрей | Львович

Плотников | Марк | Арсентьевич

Носов | Иван | Глебович

Николаев | Андрей | Матвеевич

Григорьев | Даниил | Иванович

Абрамов | Платон | Степанович

Зыков | Фёдор | Матвеевич

Аналогичный запрос можно составить, используя ключевое слово BETWEEN.

SELECT surname, name, patronymic

FROM student

WHERE birthday BETWEEN '25/06/2002' AND '25/06/2003'

* 1. Проверка на членство во множестве

С помощью ключевого слова IN возможно отобрать только те кортежи, заданный атрибут которых находится в указанном списке.

Например, следующий запрос выводит список студентов, кто родился 23 ноября 2002 года или 09 февраля 2003 года.

SELECT surname, name, patronymic

FROM student

WHERE birthday IN ('23/11/2001', '20/09/2001');

surname | name | patronymic

----------+------------+------------

Романов | Егор | Глебович

Касаткин | Константин | Миронович

Алешин | Дамир | Эмирович

(3 rows)

* 1. Поиск с использованием шаблона

Для наложения более сложных условий поиска возможно воспользоваться оператором поиска шаблонов LIKE и регулярными выражениями.

*строка* LIKE *шаблон* [ESCAPE *спецсимвол*]

*строка* NOT LIKE *шаблон* [ESCAPE *спецсимвол*]

Оператор LIKE сравнивает анализируемую строку с заданным шаблоном и в случае совпадения отбирает эту строку. Для построения шаблона используются следующие спецсимволы:

% - любая последовательность символов

\_ - строго один символ

Также возможно использовать регулярные выражения POSIX.

^ - начало строки

$ - окончание строки

\* - повторение предыдущего символа любое число раз

\ - проверка наличия указанного после \ символа

| - выбор одного из двух вариантов

~ - поиск с учетом регистра

[...] – указание класса символов

При проверке по шаблону LIKE всегда рассматривается вся строка. Поэтому, если нужно найти последовательность символов где-то в середине строки, шаблон должен начинаться и заканчиваться знаками процента.

Например, данный запрос позволяет найти всех студентов, чье имя начинается на букву ‘А’.

SELECT surname, name, patronymic

FROM student

WHERE name LIKE 'А%';

surname | name | patronymic

----------+---------+--------------

Худяков | Андрей | Львович

Николаев | Андрей | Матвеевич

Сычев | Артём | Артёмович

Николаев | Алексей | Серафимович

Кузьмин | Али | Лукич

Воронин | Андрей | Петрович

Иванов | Артём | Маркович

Алехин | Андрей | Всеволодович

Данный запрос вернет всех студентов, чье имя состоит из 5 символов и заканчивается на букву “я”.

SELECT surname, name, patronymic

FROM student

WHERE name LIKE '\_\_\_\_я';

surname | name | patronymic

------------+-------+------------

Кондрашова | Дарья | Михайловна

Коровина | Мария | Георгиевна

Петрова | София | Николаевна

Князева | Лилия | Тимофеевна

Алехина | Дарья | Артёмовна

Соколова | Мария | Романовна

(6 rows)

Рассмотрим пример с регулярными выражениями

SELECT surname, name, patronymic

FROM student

WHERE surname ~ '^[ЛМН]';

surname | name | patronymic

-----------+-----------+---------------

Морозов | Илья | Глебович

Максимов | Виктор | Фёдорович

Носов | Иван | Глебович

Николаев | Андрей | Матвеевич

Николаев | Алексей | Серафимович

Малышев | Глеб | Гордеевич

Литвинов | Иван | Александрович

Маслов | Михаил | Дмитриевич

Данный пример вернет всех студентов, чьи фамилии начинаются на буквы Л, М, Н. Символом ^ обозначается, что следующий символ будет первым в строке. В квадратных скобках указывается список допустимых символов. Аналогично можно записать через диапазон – [Л-Н]. Символ ~ указывает, что сравнение идет с учетом регистров.

Дополним выражение таким образом, чтобы отбираемые строки оканчивались на буквосочетание `ин`.

SELECT surname, name, patronymic

FROM student

WHERE surname ~ '^[ЛМН].\*ин$';

surname | name | patronymic

---------+-----------+------------

Маркин | Даниэль | Ильич

Мухин | Ростислав | Янович

(2 rows)

Символом $ указывается, что предыдущие символы будут стоять в конце строки. Т.к. между ними и начальными символами могут находиться еще любое число символов, то обозначим их с помощью ‘.\*’. В данном случае точка обозначает «любой символ», а звездочка продублирует его от 0 до любого числа раз.

Если убрать символ окончания строки $, то будет производиться поиск строк, начинающихся на буквы Л, М, Н и содержащие в себе «ин».

SELECT surname, name, patronymic

FROM student

WHERE surname ~ '^[ЛМН].\*ин';

surname | name | patronymic

----------+-----------+---------------

Литвинов | Иван | Александрович

Литвинов | Георгий | Артёмович

Маркин | Даниэль | Ильич

Мухин | Ростислав | Янович

(4 rows)

Более подробно про регулярные выражения можно прочитать в источнике [2].

* 1. Вычисляемые столбцы

На языке SQL возможно добавлять к итоговой выборке отдельные столбцы, значения которых будут вычисляться в процессе фильтрации. Для этого, к отбираемым столбцам после ключевого слова SELECT добавляется выражение, которое будет вычисляться для каждой строки.

Например, рассчитаем возраст всех студентов вуза и выведем его вместе с ФИО. Для этого добавим еще один столбец и укажем в нем функцию *age,* позволяющую вычислить разницу между двумя датами. В качестве точки отсчета выберем текущую дату, значение которой можно получить с помощью функции CURRENT\_DATE.

SELECT surname, name, patronymic, age(CURRENT\_DATE, birthday)

FROM student;

surname | name | patronymic | age

--------------+------------+----------------+--------------------------

Иванова | Есения | Ивановна | 19 years 6 mons 5 days

Федоров | Роберт | Даниилович | 19 years 6 mons 27 days

Морозов | Илья | Глебович | 19 years 10 mons 11 days

Жилин | Никита | Максимович | 20 years 2 mons 25 days

Худяков | Андрей | Львович | 20 years 5 mons 23 days

Плотников | Марк | Арсентьевич | 20 years 2 mons 17 days

Максимов | Виктор | Фёдорович | 19 years 11 mons 17 days

Баранов | Максим | Маркович | 19 years 9 mons 17 days

Для того, чтобы переименовать столбец age, воспользуемся ключевым словом AS. После него укажем новое название столбца – «Возраст».

SELECT surname, name, patronymic, age(CURRENT\_DATE, birthday) AS "Возраст"

FROM student;

surname | name | patronymic | Возраст

--------------+------------+----------------+--------------------------

Иванова | Есения | Ивановна | 19 years 6 mons 5 days

Федоров | Роберт | Даниилович | 19 years 6 mons 27 days

Морозов | Илья | Глебович | 19 years 10 mons 11 days

Жилин | Никита | Максимович | 20 years 2 mons 25 days

Худяков | Андрей | Львович | 20 years 5 mons 23 days

Плотников | Марк | Арсентьевич | 20 years 2 mons 17 days

Максимов | Виктор | Фёдорович | 19 years 11 mons 17 days

Баранов | Максим | Маркович | 19 years 9 mons 17 days

* 1. Агрегирование и группировка

Для проведения статистических вычислений в SQL существуют агрегатные функции. Данные функции принимают на вход множество значений и возвращают одно.

Основные агрегатные функции:

1. AVG: находит среднее значение
2. COUNT(\*): находит количество строк в запросе
3. SUM: находит сумму значений
4. MIN: находит наименьшее значение
5. MAX: находит наибольшее значение

Например, в данном примере рассчитывается средний оклад всех преподавателей вуза. Т.к. атрибут «Оклад» имеет тип *money*, то для применения агрегатной функции необходимо привести его к числовому значению. Для этого используется конструкция ::numeric. Более подробно о приведении типов указано в п. 1.8.2.

**SELECT AVG(salary::numeric)::numeric(10,2) AS "Average salary"**

**FROM professor;**

Average salary

----------------

66904.76

(1 row)

SELECT SUM(wage\_rate) AS "Общее число ставок"

FROM employment;

Общее число ставок

--------------------

14.10

(1 row)

Однако, если потребуется рассчитать общее число ставок для каждого из направлений, то такой подход не сработает. Перед тем, как применять агрегирующую функцию необходимо сгруппировать кортежи по определенному признаку – в данном примере, по номеру направления. Затем необходимо применить функцию SUM к каждой из получившихся групп.

Для группировки строк в SQL служит оператор GROUP BY. Данный оператор распределяет строки, образованные в результате запроса по группам, в которых значения некоторого столбца, по которому происходит группировка, являются одинаковыми. Группировку можно производить как по одному столбцу, так и по нескольким.

Дополним запрос из предыдущего примера. В данном случае сумма будет рассчитана для каждого структурного подразделения. В подразделении 1 работают три преподавателя – поэтому их ставки просуммировались и результатом стало значение 1.00.

SELECT structural\_unit\_number AS "Structural unit number", SUM(wage\_rate) AS "Wage rate sum"

FROM Employment

GROUP BY structural\_unit\_number;

Structural unit number | Wage rate sum

------------------------+---------------

9 | 1.00

3 | 2.40

5 | 2.80

4 | 2.25

6 | 1.50

2 | 1.50

7 | 1.00

1 | 5.15

8 | 1.00

(9 rows)

Рассмотрим еще один пример. В нем происходит подсчет числа сотрудников в каждом структурном подразделении.

SELECT structural\_unit\_number AS "Structural unit number", count(\*) AS "Number of employees"

FROM employment

GROUP BY structural\_unit\_number;

Structural unit number | Number of employees

------------------------+---------------------

9 | 2

3 | 4

5 | 5

4 | 9

6 | 3

2 | 6

7 | 2

1 | 9

8 | 2

(9 rows)

Для фильтрации строк перед группировкой использовалось ключевое слово WHERE. В случае, если нужно отфильтровать строки после неё – используется ключевое слово HAVING.

Добавим в приведенный выше пример условие, чтобы число сотрудников выводилось только для подразделений, в которых более 2х преподавателей.

SELECT structural\_unit\_number AS "Structural unit number", count(\*) AS "Number of employees"

FROM employment

GROUP BY structural\_unit\_number

HAVING count(\*) > 2;

Structural unit number | Number of employees

------------------------+---------------------

3 | 4

5 | 5

4 | 9

6 | 3

2 | 6

1 | 9

(6 rows)

* 1. Сортировка

Для сортировки результата запроса необходимо использовать ключевое слово ORDER BY. После него указывается атрибуты, по которым производится сортировка. Далее указывается порядок с помощью слов DESC (в порядке убывания) и ASC (в порядке возрастания). По умолчанию строки сортируются по возрастанию, поэтому ASC можно опускать.

SELECT structural\_unit\_number AS "Номер структурного подразделения", count(\*) AS "Число сотрудников"

FROM employment

GROUP BY structural\_unit\_number

ORDER BY structural\_unit\_number;

Номер структурного подразделения | Число сотрудников

----------------------------------+-------------------

1 | 9

2 | 6

3 | 4

4 | 9

5 | 5

6 | 3

7 | 2

8 | 2

9 | 2

(9 rows)

SELECT structural\_unit\_number AS "Номер структурного подразделения", count(\*) AS "Число сотрудников"

FROM employment

GROUP BY structural\_unit\_number

ORDER BY structural\_unit\_number DESC;

Номер структурного подразделения | Число сотрудников

----------------------------------+-------------------

9 | 2

8 | 2

7 | 2

6 | 3

5 | 5

4 | 9

3 | 4

2 | 6

1 | 9

(9 rows)

* 1. Оконные функции

При составлении запросов с использованием агрегирующих функций применялась группировка. Все строки с одинаковыми значениями, указанными после GROUP BY объединялись в одну и над каждой из данных групп совершалось определенное действие. Таким образом, число строк в результирующей таблице уменьшалось. Однако в некоторых случаях бывает полезным провести вычисления над группой и добавить вычисленное значение в качестве дополнительного столбца для каждой строки таблицы. Например, необходимо вывести студентов всех групп и пронумеровать в алфавитном порядке внутри каждой группы. Для этого можно воспользоваться оконными функциями.

В общем виде оконные функции выглядят следующим образом:

SELECT

Название функции (столбец для вычислений)

OVER (

PARTITION BY столбец для группировки

ORDER BY столбец для сортировки

)

В качестве функции может выступать одна из агрегатных функций (SUM, COUNT, AVG, MIN, MAX) или специальные функции, предназначенные для оконных вычислений.

Приведем некоторые из них:

**ROW\_NUMBER** – данная функция возвращает номер строки и используется для нумерации;

**FIRST\_VALUE** или **LAST\_VALUE** — с помощью функции можно получить первое и последнее значение в окне. В качестве параметра принимает столбец, значение которого необходимо вернуть;

**CUME\_DIST** — вычисляет интегральное распределение (относительное положение) значений в окне;

После ключевых слов PARTITION BY необходимо указать поле, по которому будет производиться объединение в группы. Далее возможно отсортировать значения внутри каждой из групп. [3]

В итоге запрос для вычисления порядкового номера студента будет выглядеть следующим образом:

SELECT row\_number() OVER (partition by students\_group\_number ORDER BY student.surname),

surname, name, patronymic

FROM student

Рассмотрим еще один пример, где каждому студенту добавляется поле, содержащее средний балл всей группы.

SELECT DISTINCT

surname,

name,

patronymic,

avg(field\_comprehension.mark) OVER(partition by students\_group\_number)::numeric(8,2),

student.students\_group\_number

FROM student

INNER JOIN field\_comprehension ON field\_comprehension.student\_id = student.student\_id;

* 1. Несколько важных замечаний
     1. Экранирование кавычек

При использовании текстовых строк в запросах их необходимо обрамлять одинарными кавычками. Как, например, в следующем запросе:

SELECT surname, name

FROM student

WHERE name = 'Анна'

Однако, как поступить, в случае наличия в самой строке символа кавычки. Например, в институт поступил иностранный студент с фамилией O’Brien.

SELECT surname, name

FROM student

WHERE surname = 'O'Brien'

Данный запрос выполнен не будет, т.к. СУБД распознает как строку только символ 'O'. Чтобы одинарная кавычка воспринималась как часть строки, а не указание на её окончание, в PostgreSQL предусмотрена возможность дублирования данной кавычки.

SELECT surname, name

FROM student

WHERE surname = 'O''Brien'

Способ, указанный выше, сработает.

* + 1. Типы данных в PostgreSQL

PostgreSQL предоставляет пользователям большой выбор встроенных типов данных. В таблице 1 приведены основные из них. Более подробно можно прочитать в Главе 8 документации PostgreSQL. [2]

Таблица 1 Типы данных PostgreSQL

|  |  |
| --- | --- |
| **Имя** | **Описание** |
| bigint | знаковое целое из 8 байт |
| boolean | логическое значение (true/false) |
| date | календарная дата (год, месяц, день) |
| integer | знаковое четырёхбайтное целое |
| json | текстовые данные JSON |
| money | денежная сумма |
| numeric [ (*p*, *s*) ] | вещественное число заданной точности |
| real | число одинарной точности с плавающей точкой (4 байта) |
| smallint | знаковое двухбайтное целое |
| serial | четырёхбайтное целое с автоувеличением |
| text | символьная строка переменной длины |
| time [ (*p*) ] [ without time zone ] | время суток (без часового пояса) |
| uuid | универсальный уникальный идентификатор |
| varchar(n) | строка ограниченной переменной длины |

К основным целым числовым типам можно отнести типы smallint(2 байта), integer (4 байта), bigint (8 байт). В большинстве случаем рекомендуется использовать тип integer.

Для хранения дробных чисел произвольной точности возможно использовать тип numeric.

NUMERIC(точность, масштаб)

где точность – число значащих цифр (по обе стороны запятой), масштаб – число цифр после запятой. По умолчанию, тип *numeric* может хранить числа, содержащие до 1000 значащих цифр.

Для хранения чисел с плавающей точкой используются типы real и double. Обратим внимание, что при арифметике чисел с плавающей точкой возможны ошибки округления и неточности при хранении данных, поэтому при необходимости точности вычислений рекомендуется использовать тип *numeric.*

Данная проблема может стать критичной при хранении денежных сумм. Поэтому в случае хранения денежных единиц рекомендуется использовать тип numeric или money.

Для хранения строковых типов ограниченного размера рекомендуется использовать тип *varchar*(N), где N – максимальный размер строки. При необходимости хранения длинного текста возможно применять тип *text*.

Отдельно остановимся на типе, хранящим значения даты и времени. В PostgreSQL существует множество подобных типов. Наиболее популярными являются типы *date* и *time*. Все даты в PostgreSQL считаются по Григорианскому календарю, даже для времени до его введения.

Данные о дате могут храниться разными способами. В соответствии со стандартом ISO 8601, рекомендуется хранить дату в формате «год-месяц-день». Например, 1998-08-26. Также возможно использовать следующую запись: месяц/день/год в режиме MDY или день/месяц/год в режиме DMY. Для проверки, в каком режиме работает PostgreSQL возможно выполнить команду:

SELECT current\_setting('datestyle');

current\_setting

-----------------

ISO, DMY

(1 row)

Для переключения между режимами возможно использовать команду:

SET datestyle TO "ISO, DMY";

В качестве первичного ключа очень часто используют последовательность из чисел от 1 и выше. Для хранения подобной последовательности используется тип *serial*. При создании переменной данного типа создается автоинкрементирующийся счетчик, увеличивающийся с добавлением новой записи. При вызове команды INSERT поле с данным типом возможно отметить значением DEFAULT или просто оставить пустым.

В некоторых случаях необходимо, чтобы первичным ключом случил уникальный идентификатор, не повторяющийся ни в одной другой базе данных. Это может быть полезно, при объединении двух таблиц из различных баз данных. Таким идентификатором может служить GUID - глобальный уникальный идентификатор. Для его хранения в PostgreSQL используется тип *uuid*. UUID записывается в виде последовательности шестнадцатеричных цифр, разделённых знаками минуса на несколько групп: группа из 8 цифр, три группы из 4 цифр, группа из 12 цифр, что в сумме составляет 32 цифры и представляет 128 бит.

Пример *uuid*: 47a8229b-9e8a-0473-fd20-21c889da75bf

* + 1. Преобразование типов

Преобразование типов в PostgreSQL возможно выполнить несколькими способами. Самый простой из них – напрямую указать тип данных через символ “::”

SELECT 1234::int;

int4

------

1234

(1 row)

SELECT 1234::text;

text

------

1234

(1 row)

Аналогичного результата можно добиться, используя оператор CAST

SELECT CAST(1234 AS int);

int4

------

1234

(1 row)

SELECT CAST(1234 AS text);

text

------

1234

(1 row)

1. Практическая часть
   1. Задание 1

*Исследование типов данных*

Предположим, что в магазине новый конструктор стоит 999 рублей и 99 копеек. Студент С. решил приобрести для дальнейшей перепродажи 100000 таких товаров. Для расчета общей суммы, которую необходимо заплатить был создан следующий скрипт на языке PL/pgSQL. Более подробно о нём будет рассказано в лабораторной работе 6. Обратите внимание, что значение суммы имеет тип *real*.

DO

$$

DECLARE

summ real :=0.0;

BEGIN

FOR i IN 1..100000 LOOP

summ := summ + 999.99;

END LOOP;

RAISE NOTICE 'Summ = %;', summ;

--RAISE NOTICE 'Diff = %;', 99999000.00 - summ;

END

$$ language plpgsql;

Запустите скрипт. Раскомментируйте строку с вычислением разницы и определите, сколько денег переплатил студент С? Объясните полученный результат. Измените тип суммы на *numeric* и *money*. Какой результат был получен в обоих случаях?

* 1. Задание 2

Напишите SQL запрос к учебной базе данных в соответствии с вариантом.

|  |  |
| --- | --- |
| № варианта |  |
| 1 | Вывести ФИО и дату рождения самого молодого студента |
| 2 | Выведите количество студентов, которые обучаются на третьем курсе. |
| 3 | Выведите студентов, которые моложе 20 лет |
| 4 | Выведите всех студентов, обучающихся на 2 курсе (ФИО, номер группы) (используйте регулярное выражение) |
| 5 | Найдите дисциплину, по которой зачёт получило больше 100 студентов. |
| 6 | Вывести ФИО всех студентов второго курса в порядке убывания |

* 1. Задание 3

Напишите SQL запрос к учебной базе данных в соответствии с вариантом.

|  |  |
| --- | --- |
| № варианта |  |
| 1 | Посчитать количество групп 4-го курса. |
| 2 | Выведите всех студентов, средняя оценка которых больше 3, в порядке возрастания оценки. |
| 3 | Выведите то, сколько групп ИВТ учится очно/заочно |
| 4 | Посчитайте ср. Оклад по должностям, вывести в порядке возрастания. |
| 5 | Была потеряна контрольная работа студента с инициалами А. Сергеева, но не была указано группа.  Необходимо определить группу, в которой обучается студент и вернуть работу. |
| 6 | Выведите кол-во студентов из 3 курса, чьи фамилии не заканчиваются на “а”(Вывести группу и кол-во) |

* 1. Задание 4

Напишите SQL запросы к учебной базе данных в соответствии с вариантом.

|  |  |
| --- | --- |
| № варианта |  |
| 1 | Посчитать, сколько должников и в каких группах. |
| 2 | Выведите фамилии и количество их повторений, которые начинаются на ту же букву, что и ваша фамилия. |
| 3 | Выведите кол-во групп с очной формой обучения в подразделениях, исключая 2 подразделение. |
| 4 | Вывести Номера Групп на Очном обучении и номера их структурных подразделений (больше 2), переименовать столбец Номера структурных подразделений сгруппировать их и сортировать. |
| 5 | Для оптимизации БД - нужно посчитать количество символов, используемых для названия учебных подразделений. Нужно сделать запрос, отображающий это в порядке убывания, для удобства. |
| 6 | Выведите студентов, у которых дата рождения совпадает с вашей (месяц и день) |

* 1. Задание 5

Самостоятельно разработайте 5 **осмысленных** запросов к базе данных, используя приведенные в данной лабораторной работе материалы. Вариант выбирается в соответствии с номером по списку.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Вариант 1,5,9 | | | | | Вариант 2,6,10 | | | | | Вариант 3,7 | | | | | Вариант 4,8 | | | | |
| Ключевое слово | 1 | 2 | 3 | 4 | 5 | 1 | 2 | 3 | 4 | 5 | 1 | 2 | 3 | 4 | 5 | 1 | 2 | 3 | 4 | 5 |
| WHERE | + | + |  |  | + | + | + | + | + | + | + | + | + |  | + | + | + |  | + | + |
| GROUP BY |  |  | + | + | + |  |  | + | + | + |  |  | + | + | + |  |  | + | + | + |
| HAVING |  |  |  | + | + |  |  |  |  | + |  |  |  | + | + |  |  |  |  | + |
| ORDER BY |  |  | + |  | + | + |  |  | + | + |  | + |  |  | + | + |  |  | + | + |
| AS | + |  |  |  | + |  |  | + |  | + | + |  |  |  | + |  |  | + |  | + |
| LIKE |  |  |  |  | + |  | + |  |  |  |  |  |  |  | + |  | + |  |  |  |
| Агрегатные функции |  |  | + |  |  |  |  |  |  | + |  |  |  | + |  |  |  | + |  |  |
| Регулярные выражения |  | + |  |  |  |  |  | + |  |  |  | + |  |  |  |  | + |  |  |  |

# Список литературы

|  |  |
| --- | --- |
| [1] | Документация к PostgreSQL 15.1, 2022. |
| [2] | «Исходный код СУБД postgres,» [В Интернете]. Available: https://github.com/postgres/postgres. [Дата обращения: 30 01 2023]. |
| [3] | Е. Рогов, PostgreSQL изнутри, 1-е ред., Москва: ДМК Пресс, 2023, p. 662 . |
| [4] | Б. А. Новиков, Е. А. Горшкова и Н. Г. Графеева, Основы технологии баз данных, 2-е ред., Москва: ДМК пресс, 2020, p. 582. |
| [5] | Е. П. Моргунов, PostgreSQL. Основы языка SQL, 1-е ред., Санкт-Петербург: БХВ-Петербург, 2018, p. 336. |

1. В результате данного запроса должно быть выведено 163 строчки. Для экономии места, в данном лабораторном практикуме будет оставлено не более 8 строк от результата любого запроса [↑](#footnote-ref-1)