**Лаба 3**

Запрос на подсчет количества строк в таблице, удовлетворяющих заданному условию (COUNT)```

SELECT COUNT(\*) AS КоличествоУчастников FROM Участники WHERE Рейтинг > 2400;

Запрос на подсчет среднего значения в каком-либо столбце таблицы (функция AVG)

SELECT AVG(Рейтинг) AS СреднийРейтинг FROM Участники;

Запрос на подсчет суммы значений какого-либо столбца в таблице (функция SUM)

SELECT SUM(Рейтинг) AS СуммаРейтингов FROM Участники;

Запрос на нахождение максимального значения в столбце таблицы (функция MAX)

SELECT MAX(Рейтинг) AS МаксимальныйРейтинг FROM Участники;

Запрос на нахождение минимального значения в столбце таблицы (функция MIN)

SELECT MIN(Рейтинг) AS МинимальныйРейтинг FROM Участники;

Запрос на нахождение минимального значения в столбце таблицы без использования функции MIN, применяя ORDER BY и LIMIT

SELECT Рейтинг FROM Участники ORDER BY Рейтинг ASC LIMIT 1;

Запрос на нахождение максимального значения в столбце таблицы без использования функции MAX, применяя ORDER BY DESC и LIMIT

SELECT Рейтинг FROM Участники ORDER BY Рейтинг DESC LIMIT 1;

Запрос с группировкой строк и подсчетом значения любой агрегатной функции по каждой группе (GROUP BY)

SELECT ТурнирID, COUNT(\*) AS КоличествоУчастников FROM Участники GROUP BY ТурнирID;

Запрос с соединением не менее, чем 2-х таблиц, группировкой строк и подсчетом значения любой агрегатной функции по каждой группе (GROUP BY)

SELECT t.Название, COUNT(p.ПартияID) AS КоличествоПартий

FROM Турниры t

JOIN Партии p ON t.ТурнирID = p.ТурнирID

GROUP BY t.Название;

Запрос SELECT с использованием вложенного подзапроса SELECT

SELECT Имя, Рейтинг

FROM Участники

WHERE ТурнирID = (

SELECT ТурнирID

FROM Партии

GROUP BY ТурнирID

ORDER BY COUNT(\*) DESC

LIMIT 1

);

Запрос INSERT с использованием вложенного подзапроса SELECT

INSERT INTO Участники (Имя, Рейтинг, ТурнирID)

SELECT Имя, Рейтинг, 2

FROM Участники

WHERE ТурнирID = 1;

Запрос на объединение результатов запросов с использованием UNION

SELECT Имя FROM Участники WHERE ТурнирID = 1

UNION

SELECT Имя FROM Участники WHERE ТурнирID = 2;

Запрос на соединение таблиц с использованием JOIN (все виды)

-- Внутреннее соединение (INNER JOIN)

SELECT u.Имя, t.Название

FROM Участники u

INNER JOIN Турниры t ON u.ТурнирID = t.ТурнирID;

-- Левое соединение (LEFT JOIN)

SELECT u.Имя, t.Название

FROM Участники u

LEFT JOIN Турниры t ON u.ТурнирID = t.ТурнирID;

-- Правое соединение (RIGHT JOIN)

SELECT u.Имя, t.Название

FROM Участники u

RIGHT JOIN Турниры t ON u.ТурнирID = t.ТурнирID;

-- Полное соединение (FULL JOIN)

SELECT u.Имя, t.Название

FROM Участники u

FULL JOIN Турниры t ON u.ТурнирID = t.ТурнирID;

1. Что такое агрегирующие функции и как они используются?

Агрегирующие функции — это функции, которые выполняют вычисления над набором значений и возвращают одно значение. Они часто используются для получения суммарных, средних, минимальных, максимальных значений и количества строк в наборах данных. Агрегирующие функции применяются в -запросах, особенно в сочетании с предложением GROUP BY для группировки данных.

Примеры агрегирующих функций:

* SUM: Суммирует значения в столбце.
* AVG: Вычисляет среднее значение в столбце.
* MAX: Находит максимальное значение в столбце.
* MIN: Находит минимальное значение в столбце.
* COUNT: Подсчитывает количество строк.

2. Как называются функции SUM, AVG, MAX, MIN, COUNT? Почему?

Функции SUM, AVG, MAX, MIN, COUNT называются агрегирующими функциями, потому что они выполняют операции над наборами значений и возвращают одно агрегированное значение. Названия этих функций отражают их предназначение:

* SUM (сумма): Суммирует значения.
* AVG (среднее): Вычисляет среднее значение.
* MAX (максимум): Находит максимальное значение.
* MIN (минимум): Находит минимальное значение.
* COUNT (подсчет): Подсчитывает количество строк.

3. Какие требования необходимо выполнить для объединения двух запросов?

Для объединения двух запросов с использованием UNION, INTERSECT или EXCEPT, необходимо, чтобы:

* Обе выборки имели одинаковое количество столбцов.
* Типы данных в соответствующих столбцах были совместимы.
* Порядок столбцов был одинаковым.

4. Как средствами выполнить объединение, пересечение и разность двух таблиц?

* Объединение (UNION): Возвращает уникальные строки из обоих запросов.

SELECT column1, column2 FROM table1 UNION SELECT column1, column2 FROM table2;

* Пересечение (INTERSECT): Возвращает только те строки, которые присутствуют в обоих запросах.

SELECT column1, column2 FROM table1 INTERSECT SELECT column1, column2 FROM table2;

* Разность (EXCEPT): Возвращает строки, которые присутствуют в первом запросе, но отсутствуют во втором.

SELECT column1, column2 FROM table1 EXCEPT SELECT column1, column2 FROM table2;

5. Каковы особенности применения предложения UNION?

* UNION объединяет результаты двух запросов и удаляет дублирующиеся строки.
* Для включения дублирующихся строк используется UNION ALL.
* Столбцы в обоих запросах должны иметь одинаковое количество и совместимые типы данных.
* Порядок и имена столбцов в результате будут взяты из первого запроса.

6. Какие виды JOIN реализованы в Postgre ?

В Postgre реализованы следующие виды JOIN:

* INNER JOIN: Возвращает строки, которые имеют совпадающие значения в обеих таблицах.
* LEFT JOIN (или LEFT OUTER JOIN): Возвращает все строки из левой таблицы и совпадающие строки из правой таблицы. Несовпадающие строки из правой таблицы получают значение NULL.
* RIGHT JOIN (или RIGHT OUTER JOIN): Возвращает все строки из правой таблицы и совпадающие строки из левой таблицы. Несовпадающие строки из левой таблицы получают значение NULL.
* FULL JOIN (или FULL OUTER JOIN): Возвращает все строки, когда есть совпадения в левой или правой таблице. Несовпадающие строки из обеих таблиц получают значение NULL.
* CROSS JOIN: Возвращает декартово произведение двух таблиц.
* SELF JOIN: Соединение таблицы с самой собой.

7. Каковы особенности применения предложения JOIN?

* JOIN используется для объединения строк из двух или более таблиц на основе связанного столбца.
* **INNER JOIN** возвращает только совпадающие строки.
* **LEFT JOIN** и **RIGHT JOIN** возвращают все строки из одной таблицы и совпадающие строки из другой таблицы, заполняя NULL для несовпадающих строк.
* **FULL JOIN** возвращает все строки, когда есть совпадения в любой из таблиц.
* **CROSS JOIN** возвращает декартово произведение строк таблиц, что может привести к значительному увеличению количества строк в результате.

**8. Для чего предназначена конструкция WITH?**

Конструкция **WITH** используется для создания временных именованных подзапросов, которые можно использовать в основном запросе. Она улучшает читаемость и упрощает управление сложными запросами.

Пример использования **WITH**:

WITH Подзапрос AS ( SELECT column1, column2 FROM table1 WHERE condition ) SELECT column1, column2 FROM Подзапрос WHERE other\_condition;

Конструкция **WITH** полезна для:

* Разделения сложных запросов на более управляемые части.
* Повышения читаемости и удобства сопровождения кода.
* Переиспользования подзапросов в основном запросе.

**Лаба 4**

1. Представления

Свойства:

Представление в базе данных - это виртуальная таблица, которая основана на результатах выполнения SQL-запроса.

Представления можно рассматривать как фильтры для данных, позволяющие предоставлять только необходимую информацию пользователям или приложениям.

Они могут объединять данные из разных таблиц или скрывать сложные запросы за простым интерфейсом.

Преимущества использования:

Улучшение безопасности: представления могут скрывать определенные данные от пользователей, позволяя им видеть только то, что им разрешено.

Упрощение доступа к данным: представления предоставляют удобный способ просмотра данных без необходимости писать сложные SQL-запросы каждый раз.

2. Хранимые процедуры

Свойства:

Хранимая процедура - это набор предварительно скомпилированных инструкций SQL, хранящихся в базе данных.

Они могут принимать параметры и возвращать результаты.

Хранимые процедуры обычно выполняются на сервере базы данных, что может уменьшить нагрузку на сеть.

Преимущества перед прямыми SQL-запросами:

Улучшенная производительность: хранимые процедуры могут выполняться быстрее, чем эквивалентные запросы, поскольку они компилируются и кэшируются на сервере.

Повторное использование кода: хранимые процедуры можно вызывать из различных частей приложения, что способствует повторному использованию кода и уменьшает дублирование.

3. Функции

Свойства:

Функция в базе данных - это фрагмент программного кода, который принимает аргументы, выполняет определенные действия и возвращает результат.

Они могут использоваться для вычислений, преобразования данных или выполнения сложных операций.

Функции бывают скалярными (возвращающими одно значение) и множественными строками (возвращающими набор строк).

Преимущества:

Уменьшение дублирования кода: функции позволяют избежать повторения одних и тех же операций в различных частях приложения.

Упрощение сложных запросов: функции могут использоваться для инкапсуляции сложной логики запросов, делая их более понятными и легкими для поддержки.

4. Триггеры

Свойства:

Триггер в базе данных - это специальный тип хранимого объекта, который автоматически запускается при определенном событии (например, при вставке, обновлении или удалении строк из таблицы).

Они позволяют реализовывать бизнес-логику на уровне базы данных, обеспечивая целостность данных и автоматизируя определенные действия.

Триггеры бывают BEFORE (до выполнения операции), AFTER (после выполнения операции) и INSTEAD OF (вместо выполнения операции).

Преимущества:

Гарантия целостности данных: триггеры могут применять ограничения или выполнить определенные действия перед или после изменения данных, обеспечивая их целостность.

Автоматизация бизнес-процессов: они позволяют автоматизировать определенные задачи или бизнес-процессы, что уменьшает вероятность ошибок и упрощает управление данными.

**Лаба 5**