

1. Select

Для выборки данных из базы данных в PostgreSQL применяется команда `SELECT`. В упрощенном виде она имеет следующий синтаксис:

```
SELECT список_столбцов FROM имя_таблицы;
```

Например, пусть ранее была создана таблица `geese`, и в нее добавлены некоторые начальные данные:

```
CREATE TABLE geese (  
    goose_id SERIAL PRIMARY KEY,  
    name VARCHAR(100),  
    breed VARCHAR(100),  
    age INT,  
    weight DECIMAL(5, 2),  
    date_of_birth DATE  
);  
  
INSERT INTO geese (name, breed, age, weight, date_of_birth)  
VALUES  
( 'Гусь 1', 'Embden', 3, 5.2, '2021-05-20'),  
( 'Гусь 2', 'Toulouse', 2, 4.8, '2022-03-15'),  
( 'Гусь 3', 'African', 1, 4.3, '2023-01-10');
```

Получим все объекты из этой таблицы:

```
SELECT * FROM geese;
```

Символ звездочка `*` указывает, что нам нужно получить все столбцы.

Команда `SELECT` и получение данных в PostgreSQL

Стоит отметить, что использование символа `*` для получения данных считается не очень хорошей практикой, так как обычно требуется получить данные только по небольшому набору столбцов. Более оптимальный подход заключается в указании всех необходимых столбцов после слова `SELECT`. Исключение составляют случаи, когда нужно получить данные по абсолютно всем столбцам таблицы или когда названия столбцов неизвестны.

Если необходимо получить данные не из всех, а только из некоторых столбцов, их имена перечисляются через запятую после `SELECT` :

```
SELECT name, breed FROM geese;
```

Выборка данных с использованием вычислений

Спецификация столбца необязательно должна представлять его точное название. Это может быть любое выражение, например, результат арифметической операции. Так, выполним следующий запрос:

```
SELECT name, weight * age  
FROM geese;
```

Здесь при выборке будут созданы два столбца. Причем второй столбец представляет значение столбца `weight`, умноженное на значение столбца `age`, что можно использовать для вычислений, таких как индекс массы гуся.

Псевдонимы столбцов и оператор AS

С помощью оператора `AS` можно изменить название выходного столбца или определить его псевдоним:

```
SELECT name AS GooseName, weight * age AS WeightAgeIndex  
FROM geese;
```

Здесь для первого столбца устанавливается псевдоним `GooseName`, хотя он представляет столбец `name`. Второй столбец, `WeightAgeIndex`, хранит произведение столбцов `weight` и `age`.

Пример использования команды SELECT

1 `select * from geese`

Data Output Messages Notifications

≡

📄

▼

📋

▼

🗑️

🗄️

⬇️

📈

SQL

	goose_id [PK] integer	name character varying (100)	breed character varying (100)	age integer	weight numeric (5,2)	date_of_birth date
1	1	Гусь 1	Белый	3	5.20	2021-04-12
2	2	Гусь 2	Черный	4	6.10	2020-03-25
3	3	Гусь 3	Белый	2	4.80	2022-05-15
4	4	Гусь 4	Черный	5	7.30	2019-08-02
5	5	Гусь 5	Серый	1	3.60	2023-06-22



2. Where

Для выборки данных из базы данных с фильтрацией в PostgreSQL также используется оператор `WHERE`. После слова `WHERE` указывается условие, которое должно быть выполнено для того, чтобы строка попала в результирующую выборку.

Операторы сравнения

В PostgreSQL, как и в MySQL, можно использовать различные операторы сравнения:

- `=` : равенство
- `≠` или `<>` : неравенство
- `<` : меньше чем
- `>` : больше чем
- `≤` : меньше или равно
- `≥` : больше или равно

Пример 1: Выборка гусей по породе

Например, чтобы выбрать всех гусей, порода которых — `Embden`, можно использовать следующий запрос:

```
SELECT * FROM geese
WHERE breed = 'Embden';
```

Пример 2: Выборка по возрасту

Для того чтобы выбрать всех гусей, чей возраст больше 2 лет, запрос будет следующим:

```
SELECT * FROM geese
WHERE age > 2;
```

Пример 3: Совокупная стоимость больше 100

Если нужно выбрать всех владельцев, у которых совокупный вес гусей больше 100 кг, можно выполнить запрос, используя арифметическую операцию:

```
SELECT * FROM geese
WHERE weight * age > 100;
```

Логические операторы

Логические операторы в PostgreSQL позволяют комбинировать несколько условий. Вот список доступных операторов:

- **AND**: логическое И — все условия должны быть истинными.
- **OR**: логическое ИЛИ — хотя бы одно условие должно быть истинным.
- **NOT**: логическое отрицание — инвертирует результат выражения.

Пример 4: Логическое И (AND)

Выберем всех владельцев, которые имеют гусей породы `Toulouse` и возраст которых больше 1 года:

```
SELECT * FROM geese
WHERE breed = 'Toulouse' AND age > 1;
```

Пример 5: Логическое ИЛИ (OR)

Теперь выберем всех владельцев, которые имеют гусей породы Toulouse или чей возраст больше 2 лет:

```
SELECT * FROM geese
WHERE breed = 'Toulouse' OR age > 2;
```

Пример 6: Логическое НЕ (NOT)

Если необходимо выбрать всех гусей, которые не имеют породу Toulouse, используем оператор NOT:

```
SELECT * FROM geese
WHERE NOT breed = 'Toulouse';
```

Приоритет операций

В одном условии можно комбинировать несколько логических операций. В PostgreSQL приоритет операций следующий:

1. NOT (отрицание)
2. AND (логическое И)
3. OR (логическое ИЛИ)

Query

Query History

1

SELECT * FROM geese

2

WHERE breed = 'Toulouse' OR age > 2;

Data Output

Messages

Notifications

≡

+

📄

▼

📋

▼

🗑️

🗑️

📄

📄

📄

📄

📄

📄

SQL

	goose_id [PK] integer	name character varying (100)	breed character varying (100)	age integer	weight numeric (5,2)	date_of_birth date
1	1	Гусь 1	Белый	3	5.20	2021-04-12
2	2	Гусь 2	Черный	4	6.10	2020-03-25
3	4	Гусь 4	Черный	5	7.30	2019-08-02

```
1 SELECT * FROM Products
2 WHERE Manufacturer = 'Apple';
```

Data Output Explain Messages Query History					
	id	productname	manufacturer	productcount	price
	integer	character varying (30)	character varying (20)	integer	numeric
1	1	iPhone X	Apple	3	36000
2	2	iPhone 8	Apple	2	41000

3. HAVING

HAVING позволяет фильтровать агрегированные группы после группировки. Это особенно полезно для фильтрации групп по результатам агрегатных функций.

Пример:

Чтобы узнать, в каких породах гусей больше одного представителя, используем **HAVING** для фильтрации групп, где количество меньше двух:

```
SELECT breed, COUNT(*) AS GeeseCount
FROM geese
GROUP BY breed
HAVING COUNT(*) > 1;
```

Этот запрос показывает только те породы, в которых есть более одного гуся.

Использование **WHERE** и **HAVING** вместе

WHERE и **HAVING** можно комбинировать для фильтрации как отдельных строк, так и агрегированных групп. Сначала **WHERE** фильтрует строки по указанным условиям, затем **GROUP BY** создает группы, а **HAVING** применяет фильтрацию к агрегированным данным.

Пример:

Чтобы выбрать породы гусей, где каждый гусь весит более 5 кг, и оставить только те породы, где таких гусей больше одного:

```
SELECT breed, COUNT(*) AS HeavyGeeseCount
FROM geese
WHERE weight > 5
```

```
GROUP BY breed  
HAVING COUNT(*) > 1;
```

Здесь `HeavyGeeseCount` показывает количество гусей в каждой породе, у которых вес больше 5 кг, но запрос возвращает только те породы, где таких гусей более одного.



PostgreSQL HAVING

Query

Query History

1

2

3

4

5

SELECT breed, COUNT(*) AS HeavyGeeseCount

FROM geese

WHERE weight > 5

GROUP BY breed

HAVING COUNT(*) > 1;

Data Output

Messages

Notifications

≡+

📄

▼

📋

▼

🗑️

🗄️

⬇️

📈

SQL

	breed character varying (100) 🔒	heavygeesecount bigint 🔒
1	Черный	2

4. Between

Операторы `IN` и `BETWEEN` в SQL используются для фильтрации данных по множеству значений или диапазону. Рассмотрим их использование в контексте базы данных о гусях и их владельцах.

Оператор `IN`

Оператор `IN` позволяет выбирать строки, значения в которых соответствуют одному из элементов в списке. Этот список может быть задан вручную или динамически вычисляться с помощью подзапроса.

Синтаксис:

```
WHERE выражение [NOT] IN (значение1, значение2, ...)
```

- `IN` выбирает строки, где выражение равно одному из значений в списке.
- `NOT IN` выбирает строки, где выражение не равно ни одному из значений в списке.

Пример 1: Выбор всех гусей с определенными породами

Если нужно выбрать всех гусей определенных пород, например, "Лебедев" и "Кавказский", можно использовать оператор `IN`:

```
SELECT * FROM geese  
WHERE breed IN ('Лебедев', 'Кавказский');
```

Этот запрос вернет всех гусей, породы которых либо "Лебедев", либо "Кавказский".

Пример 2: Выбор всех владельцев, которые не имеют определенных гусей

Если нужно выбрать всех владельцев, которые не имеют гусей породы "Лебедев" или "Кавказский", используем `NOT IN`:

```
SELECT * FROM owners  
WHERE owner_id NOT IN (SELECT owner_id FROM goose_owners WHERE goose_id IN  
(SELECT goose_id FROM geese WHERE breed IN ('Лебедев', 'Кавказский')));
```

Этот запрос выберет всех владельцев, которые не имеют гусей указанных пород.

Оператор `BETWEEN`

Оператор `BETWEEN` используется для фильтрации значений в диапазоне между двумя крайними значениями. Оператор включает оба конца диапазона.

Синтаксис:

```
WHERE выражение [NOT] BETWEEN начальное_значение AND конечное_значение
```

- `BETWEEN` выбирает значения, находящиеся в указанном диапазоне, включая начальное и конечное значения.
- `NOT BETWEEN` выбирает значения, которые не входят в указанный диапазон.

Пример 1: Выбор всех гусей в определенном возрастном диапазоне

Чтобы выбрать всех гусей в возрасте от 2 до 5 лет, используем оператор `BETWEEN`:

```
SELECT * FROM geese  
WHERE age BETWEEN 2 AND 5;
```

Этот запрос вернет всех гусей, чей возраст находится между 2 и 5 годами, включая 2 и 5 лет.

Пример 2: Выбор всех посещений, которые произошли в определенный период времени

Предположим, что нам нужно выбрать все визиты к ветеринару, которые произошли в период с 2022 по 2023 год:

```
SELECT * FROM vet_visits  
WHERE visit_date BETWEEN '2022-01-01' AND '2023-12-31';
```

Этот запрос вернет все визиты, сделанные в 2022 и 2023 годах.

Пример 3: Выбор всех гусей с весом в определенном диапазоне

Чтобы выбрать всех гусей с весом от 3 до 5 кг:

```
SELECT * FROM geese  
WHERE weight BETWEEN 3 AND 5;
```

Этот запрос вернет всех гусей с весом от 3 до 5 кг, включая оба значения.

```
1 SELECT * FROM geese
2 WHERE weight BETWEEN 3 AND 5;
```

Data Output Messages Notifications

	goose_id [PK] integer	name character varying (100)	breed character varying (100)	age integer	weight numeric (5,2)	date_of_birth date
1	3	Гусь 3	Белый	2	4.80	2022-05-15
2	5	Гусь 5	Серый	1	3.60	2023-06-22

Query Editor

```
1 SELECT *
2 FROM Price
3 WHERE price NOT BETWEEN 200 AND 280;
4
```

Data Output Explain Messages Notifications

	id [PK] integer	price double precision
1	4	190
2	3	300

5. GROUP BY

GROUP BY группирует строки с одинаковыми значениями в указанных столбцах, позволяя применять агрегатные функции, такие как **COUNT**, **SUM**, **AVG** и другие. Это удобно для получения агрегированных данных по определенным критериям.

Пример:

Чтобы узнать, сколько яиц отложил каждый гусь, можно сгруппировать данные по **goose_id** и посчитать количество яиц для каждого гуся:

```
SELECT goose_id, COUNT(*) AS EggCount
FROM eggs
GROUP BY goose_id;
```

Здесь `goose_id` — это параметр группировки, а `EggCount` показывает количество яиц, отложенных каждым гусем.

Группировка по нескольким столбцам

`GROUP BY` может также выполнять группировку по нескольким столбцам, чтобы получить уникальные сочетания значений в этих столбцах.

Пример:

Чтобы узнать, сколько яиц каждого цвета отложил каждый гусь, можно сгруппировать данные по `goose_id` и `egg_color`:

```
SELECT goose_id, egg_color, COUNT(*) AS EggCount
FROM eggs
GROUP BY goose_id, egg_color;
```

Этот запрос покажет, сколько яиц определенного цвета отложил каждый гусь.

Порядок выполнения `GROUP BY` с другими операторами

Запросы с `GROUP BY` могут включать операторы `WHERE` и `HAVING` в определенном порядке:

1. `WHERE` — фильтрует строки перед группировкой.
2. `GROUP BY` — объединяет строки в группы.
3. `HAVING` — фильтрует группы.

Пример:

Чтобы выбрать только тех гусей, чей вес превышает 5 кг, и сгруппировать их по породе для подсчета количества, можно использовать `WHERE` перед `GROUP BY`:

```
SELECT breed, COUNT(*) AS HeavyGeeseCount
FROM geese
WHERE weight > 5
GROUP BY breed;
```

Здесь `HeavyGeeseCount` показывает количество гусей с массой более 5 кг для каждой породы.

Query

Query History

1

2

3

4

SELECT

breed,

COUNT(*)

AS

HeavyGeeseCount

FROM

geese

WHERE

weight >

5

GROUP BY

breed;

Data Output

Messages

Notifications

≡+

📄

▼

📋

▼

🗑️

🗄️

⬇️

📈

SQL

	breed character varying (100) 🔒	heavygeesecount bigint 🔒
1	Черный	2
2	Белый	1

6. Order

Оператор `ORDER BY`

Оператор `ORDER BY` позволяет сортировать строки по одному или нескольким столбцам. По умолчанию данные сортируются по возрастанию, но с помощью оператора `DESC` можно задать сортировку по убыванию.

Пример 1: Упорядочение по одному столбцу

Чтобы упорядочить строки в таблице `geese` по столбцу `weight`, используем следующий запрос:

```
SELECT * FROM geese
ORDER BY weight;
```

Этот запрос отсортирует данные по весу гуся в порядке возрастания.

Пример 2: Сортировка по псевдониму столбца

Также можно сортировать по значению, определенному с помощью оператора `AS`. Например, чтобы упорядочить гусей по возрасту в месяцах, можно использовать псевдоним:

```
SELECT name, age * 12 AS AgeInMonths
FROM geese
ORDER BY AgeInMonths;
```

Этот запрос создаст столбец `AgeInMonths`, который затем используется для сортировки.

Пример 3: Сложные выражения для сортировки

Сортировку также можно осуществить по сложному выражению. Например, чтобы упорядочить гусей по совокупной массе яиц, умноженной на возраст, используем такой запрос:

```
SELECT g.name, g.age, g.weight, e.egg_weight * g.age AS EggWeightByAge
FROM geese g
JOIN eggs e ON g.id = e.goose_id
ORDER BY EggWeightByAge;
```

Сортировка по убыванию

Чтобы отсортировать данные по убыванию, добавьте `DESC` после столбца. Например, для сортировки гусей по весу от самого большого к меньшему:

```
SELECT name, weight
FROM geese
ORDER BY weight DESC;
```

Сортировка по возрастанию

Сортировка по возрастанию задается ключевым словом `ASC`, хотя оно используется по умолчанию. Для сортировки гусей по возрасту в порядке возрастания:

```
SELECT name, age
FROM geese
ORDER BY age ASC;
```

Сортировка по нескольким столбцам

Для сортировки сразу по нескольким столбцам их нужно перечислить через запятую. Сначала будет использоваться первый столбец, а при равенстве значений — следующий.

```
SELECT name, weight, age
FROM geese
ORDER BY weight DESC, age ASC;
```

Этот запрос сначала сортирует по `weight` в порядке убывания, а затем, если веса совпадают, по `age` в порядке возрастания.

Query Query History

1 SELECT name, weight, age

2 FROM geese

3 ORDER BY weight DESC, age ASC;

Data Output Messages Notifications

≡+

SQL

	name character varying (100)	weight numeric (5,2)	age integer
1	Гусь 4	7.30	5
2	Гусь 2	6.10	4
3	Гусь 1	5.20	3
4	Гусь 3	4.80	2
5	Гусь 5	3.60	1

7. Limit

Операторы LIMIT и OFFSET

Операторы `LIMIT` и `OFFSET` используются для ограничения количества строк, возвращаемых в результатах запроса. Они особенно полезны при реализации分页ного вывода данных, где необходимо контролировать, сколько строк будет отображено на каждой странице.

Оператор LIMIT

Оператор `LIMIT` ограничивает количество строк, которые возвращаются в результате запроса.

Синтаксис:

```
SELECT * FROM таблица
ORDER BY столбец
LIMIT количество_строк;
```


Этот запрос вернет заданное количество строк из таблицы, упорядоченных по указанному столбцу.

Пример 1: Ограничение выборки

Если нужно выбрать только 4 строки из таблицы `geese`, можно использовать запрос:

```
SELECT * FROM geese
ORDER BY name
LIMIT 4;
```











Этот запрос вернет первые 4 строки в таблице `geese`, отсортированные по имени гуся.







1  SELECT * FROM geese

2 ORDER BY name

3 LIMIT 4;

Data Output Messages Notifications



	goose_id [PK] integer 	name character varying (100) 	breed character varying (100) 	age integer 	weight numeric (5,2) 	date_of_birth date 
1	1	Гусь 1	Белый	3	5.20	2021-04-12
2	2	Гусь 2	Черный	4	6.10	2020-03-25
3	3	Гусь 3	Белый	2	4.80	2022-05-15
4	4	Гусь 4	Черный	5	7.30	2019-08-02

Оператор OFFSET

Оператор `OFFSET` позволяет указать, с какой строки начинать выборку. Это полезно, например, при реализации страничного вывода данных.

Синтаксис:

```
SELECT * FROM таблица
ORDER BY столбец
LIMIT количество_строк OFFSET начало_с_какой_строки;
```

Пример 2: Использование LIMIT и OFFSET

Если нужно выбрать 3 строки, начиная со 2-й строки (то есть пропустить первую строку), можно использовать такой запрос:

```
SELECT * FROM geese
ORDER BY name
LIMIT 3 OFFSET 1;
```

Этот запрос пропустит первую строку и вернет следующие три строки из таблицы `geese`, отсортированные по имени гуся.

1 SELECT * FROM geese
2 ORDER BY name
3 LIMIT 3 OFFSET 1;

Data Output Messages Notifications

	goose_id [PK] integer	name character varying (100)	breed character varying (100)	age integer	weight numeric (5,2)	date_of_birth date
1	2	Гусь 2	Черный	4	6.10	2020-03-25
2	3	Гусь 3	Белый	2	4.80	2022-05-15
3	4	Гусь 4	Черный	5	7.30	2019-08-02

Комбинирование LIMIT и OFFSET

Когда требуется выбрать данные с определенной строки и ограничить выборку количеством строк, можно комбинировать оба оператора.

Пример 3: Пропуск строк и ограничение количества

Если необходимо выбрать все строки, начиная с третьей, можно использовать оператор `OFFSET` без `LIMIT`:

```
SELECT * FROM geese
ORDER BY name
```



```
OFFSET 2;
```

Этот запрос пропустит первые две строки и вернет все остальные строки, начиная с третьей.

Использование ключевого слова `ALL`

Если необходимо вернуть все строки начиная с определенной позиции, можно использовать ключевое слово `ALL` после `LIMIT`:

```
SELECT * FROM geese  
ORDER BY name  
LIMIT ALL OFFSET 2;
```

Этот запрос вернет все строки, начиная с третьей, и не будет ограничивать их количество, поскольку `LIMIT ALL` не накладывает ограничений.

8. Union

Оператор `UNION`

`UNION` используется для объединения результатов двух или более запросов `SELECT`. Запросы могут обращаться как к одной таблице, так и к разным таблицам, если у них одинаковое количество столбцов и совместимые типы данных. По умолчанию `UNION` удаляет дублирующиеся строки, но можно сохранить все строки, включая повторяющиеся, с помощью `UNION ALL`.

Синтаксис `UNION`

```
SELECT выражение1  
UNION [ALL] SELECT выражение2  
[UNION [ALL] SELECT выражениеN];
```

Примеры использования `UNION` с данными вашей базы данных

Предположим, у вас есть таблицы `geese` и `owners`, и вы хотите получить уникальный список всех имен гусей и владельцев.

Пример: Уникальные имена гусей и владельцев

```
SELECT name AS IndividualName  
FROM geese
```

```
UNION SELECT first_name AS IndividualName
FROM owners;
```

Этот запрос возвращает уникальные имена из таблицы `geese` и таблицы `owners`. Мы используем `UNION`, чтобы исключить повторяющиеся имена.

Пример: Включение всех повторяющихся строк с `UNION ALL`

Если нужно оставить все строки, включая дублирующиеся, используйте `UNION ALL`.

```
SELECT name AS IndividualName
FROM geese
UNION ALL SELECT first_name AS IndividualName
FROM owners;
```

Объединение выборок из одной таблицы с разными условиями

Допустим, в таблице `geese` мы хотим разделить гусей по возрасту: если возраст меньше 2 лет, гусю присваивается статус "Молодой", если 2 года и более — "Взрослый".

```
SELECT name, age, 'Молодой' AS Status
FROM geese
WHERE age < 2
UNION
SELECT name, age, 'Взрослый' AS Status
FROM geese
WHERE age ≥ 2;
```

Этот запрос возвращает список всех гусей с указанием их статуса в зависимости от возраста.

Сортировка после объединения

Сортировать результаты объединения можно, указав `ORDER BY` в конце объединенного запроса. Сортировка ориентируется на названия столбцов первой выборки:

```
SELECT name AS IndividualName
FROM geese
UNION SELECT first_name AS IndividualName
FROM owners
ORDER BY IndividualName;
```

Использование `UNION` с агрегацией данных

Допустим, вы хотите получить общий вес яиц и общий вес гусей в одной выборке.

```
SELECT 'Eggs' AS Source, SUM(egg_weight) AS TotalWeight
FROM eggs
UNION
SELECT 'Geese' AS Source, SUM(weight) AS TotalWeight
FROM geese;
```

Этот запрос объединяет суммарный вес всех яиц и всех гусей, добавляя к каждой строке источник данных: "Eggs" или "Geese".

Query

Query History

1

2

3

4

5

SELECT name AS IndividualName

FROM geese

UNION SELECT first_name AS IndividualName

FROM owners

ORDER BY IndividualName;

Data Output

Messages

Notifications

≡+

▼

▼

SQL

individualname

character varying (100)

🔒

1	Гусь 1
2	Гусь 2
3	Гусь 3
4	Гусь 4
5	Гусь 5
6	Иван
7	Петр
8	Сергей

9. Intersect

Оператор INTERSECT

Оператор `INTERSECT` позволяет найти общие строки между двумя выборками. Это значит, что в итоговом наборе данных останутся только те строки, которые присутствуют в обеих выборках. `INTERSECT` часто используется для нахождения пересечений данных из разных

таблиц или выборок, когда нужно найти элементы, удовлетворяющие условиям обоих запросов.

Синтаксис INTERSECT

```
SELECT выражение1  
INTERSECT  
SELECT выражение2;
```

Примеры использования INTERSECT с данными вашей базы данных

Предположим, у вас есть таблицы `geese` и `owners`, и вы хотите найти тех владельцев, чьи имена совпадают с именами гусей (возможно, владельцы назвали гусей в свою честь).

Пример: Совпадающие имена гусей и владельцев

```
SELECT name AS IndividualName  
FROM geese  
INTERSECT  
SELECT first_name AS IndividualName  
FROM owners;
```

Этот запрос возвращает список уникальных имен, которые встречаются и в таблице `geese` (как имена гусей), и в таблице `owners` (как имена владельцев).

Пример: Пересечение по критерию возраста

Допустим, вы хотите найти владельцев и гусей, чей возраст совпадает. В этом случае вам нужно будет добавить столбец возраста как критерий пересечения. Предположим, у вас есть таблица `goose_owners`, связывающая гусей и владельцев по `goose_id` и `owner_id`.

```
SELECT owners.first_name AS Name, owners.contact_info, goose_owners.goose_id  
FROM owners  
JOIN goose_owners ON owners.id = goose_owners.owner_id  
INTERSECT  
SELECT geese.name AS Name, geese.date_of_birth AS DateOfBirth,  
goose_owners.goose_id  
FROM geese  
JOIN goose_owners ON geese.id = goose_owners.goose_id;
```

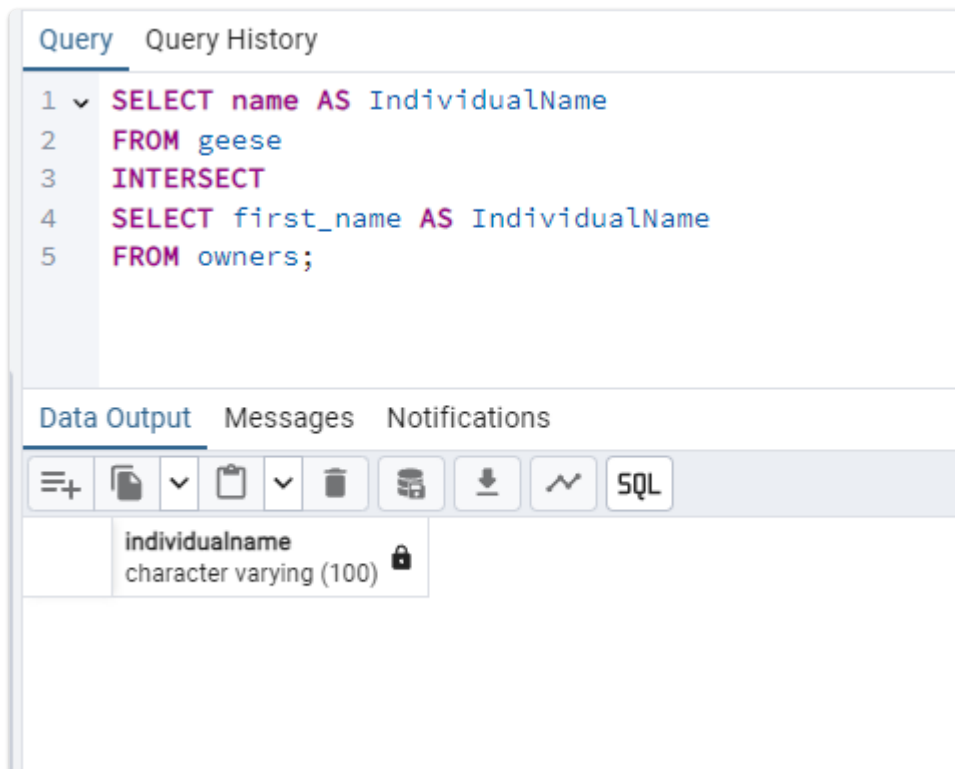
Этот запрос возвращает информацию о владельцах и гусей, у которых совпадают определенные критерии, такие как `goose_id`.

Пример: Найти совпадение по весу яйца и весу гуся

Представим, что вам нужно найти записи, где вес яйца совпадает с весом гуся.

```
SELECT geese.name AS IndividualName, weight AS Weight
FROM geese
INTERSECT
SELECT eggs.goose_id AS GooseID, egg_weight AS Weight
FROM eggs;
```

Здесь `INTERSECT` находит записи, где вес яйца совпадает с весом гуся.



10. Эксепт

Оператор EXCEPT

Оператор `EXCEPT` используется для исключения строк одной выборки из другой. Он возвращает строки из первой выборки, которые отсутствуют во второй. Это полезно для получения данных, которые есть в одной таблице, но не существуют в другой, например, для нахождения элементов, которые не имеют соответствующих записей в другой таблице.

Синтаксис EXCEPT

```
SELECT выражение1  
EXCEPT  
SELECT выражение2;
```

Примеры использования EXCEPT с данными вашей базы данных

Предположим, что в базе данных есть таблицы `geese`, `owners`, и `goose_owners`. Мы можем использовать `EXCEPT`, чтобы найти записи, которые есть в одной таблице, но отсутствуют в другой.

Пример: Найти гусей без владельцев

Допустим, вы хотите найти всех гусей, которые не имеют владельцев. В этом случае можно выполнить запрос, который находит гусей, чьи `id` отсутствуют в таблице `goose_owners`.

```
SELECT goose_id, name  
FROM geese  
EXCEPT  
SELECT goose_id, NULL  
FROM goose_owners;
```

Этот запрос возвращает всех гусей, чьи `id` отсутствуют в таблице `goose_owners`, то есть тех, кто не имеет записи о владельце.

Пример: Найти владельцев, которые не владеют ни одним гусем

Теперь предположим, что вы хотите узнать, кто из владельцев еще не закреплен ни за одним гусем.

```
SELECT owner_id, first_name, last_name  
FROM owners  
EXCEPT  
SELECT owner_id, NULL, NULL  
FROM goose_owners;
```

Этот запрос возвращает список владельцев, чьи `id` отсутствуют в таблице `goose_owners`, то есть тех, кто не привязан к гусю.

Пример: Найти владельцев, которые не использовали определенный вид корма

Допустим, вам нужно найти владельцев, чьи гуси не получали определенный тип корма, например, корм с именем "Premium Feed".

```

SELECT owner_id
FROM goose_owners
EXCEPT
SELECT goose_owners.owner_id
FROM goose_owners
JOIN feeding_schedule ON goose_owners.goose_id = feeding_schedule.goose_id
JOIN feed ON feeding_schedule.feed_id = feed.id
WHERE feed.feed_name = 'Premium Feed';

```

Этот запрос возвращает `owner_id` владельцев, чьи гуси не получали корм с названием "Premium Feed".

The screenshot shows a database client interface with a query editor and a results pane. The query editor contains the following SQL code:

```

1 SELECT goose_id, name
2 FROM geese
3 EXCEPT
4 SELECT goose_id, NULL
5 FROM goose_owners;

```

The results pane shows the output of the query. It has tabs for "Data Output", "Messages", and "Notifications". The "Data Output" tab is active, displaying a table with two columns: "goose_id" (integer) and "name" (character varying). The table contains five rows of data:

	goose_id integer	name character varying
1	2	Гусь 2
2	5	Гусь 5
3	3	Гусь 3
4	4	Гусь 4
5	1	Гусь 1

11. Distinct

Оператор `DISTINCT` в SQL используется для выборки уникальных значений из таблицы, устраняя дублирующие строки в результатах запроса. Он применяется для того, чтобы в итоговом наборе данных не повторялись одинаковые значения в выбранных столбцах.

Синтаксис:

```

SELECT DISTINCT столбец1, столбец2, ...
FROM имя_таблицы;

```

Этот запрос вернет уникальные комбинации значений из указанных столбцов.

Пример 1: Уникальные породы гусей

Допустим, у нас есть таблица `geese`, и мы хотим получить список уникальных пород гусей. Запрос будет следующим:

```
SELECT DISTINCT breed
FROM geese;
```

Этот запрос вернет список всех уникальных пород гусей, без повторений.

Пример 2: Уникальные владельцы по гусю

Если необходимо получить уникальных владельцев для каждого гуся, то можно использовать запрос с несколькими столбцами:

```
SELECT DISTINCT owner_id, goose_id
FROM goose_owners;
```

Этот запрос вернет все уникальные комбинации владельцев и их гусей.

Пример 3: Уникальные цвета яиц

Предположим, мы хотим узнать все уникальные цвета яиц, которые были снесены гусями. Запрос будет следующим:

```
SELECT DISTINCT egg_color
FROM eggs;
```

Этот запрос вернет список уникальных цветов яиц, без повторений.

Примечания:

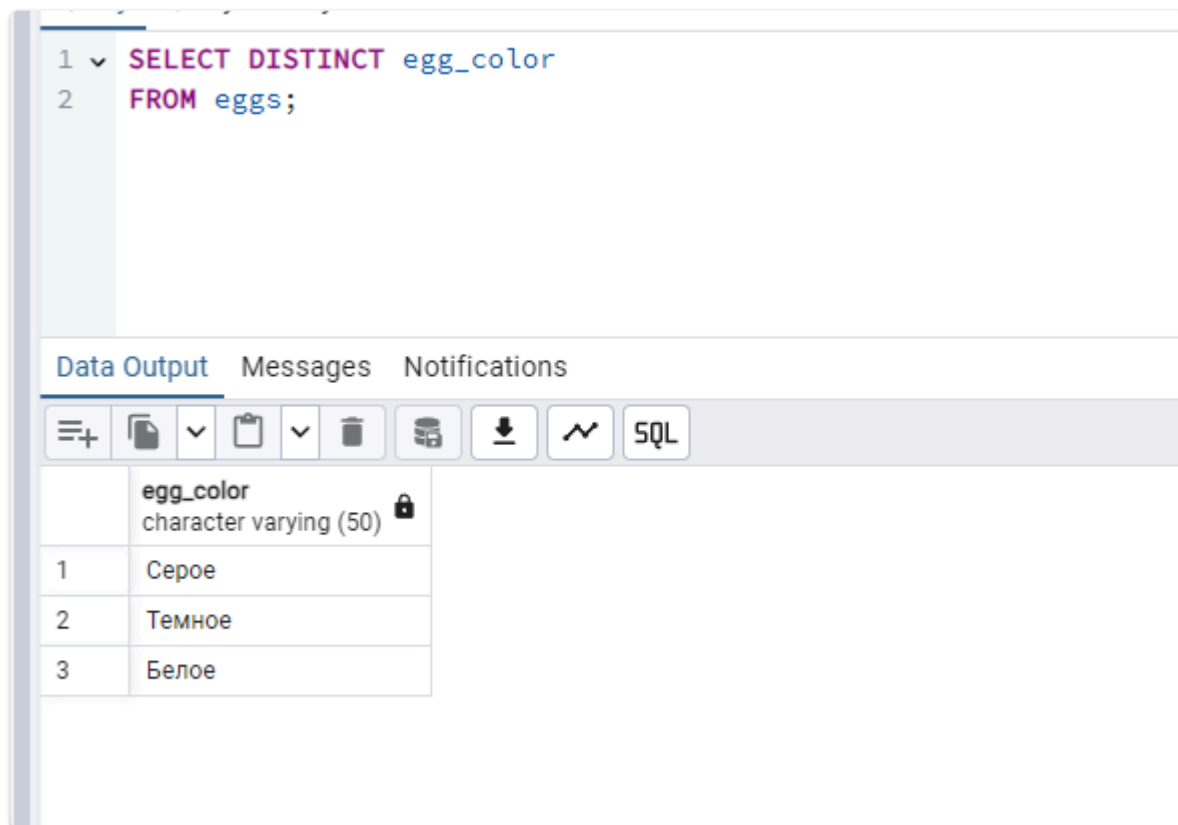
- Оператор `DISTINCT` применяет уникальность ко всем столбцам, указанным в запросе. Это означает, что если в запросе указаны несколько столбцов, то комбинация всех этих столбцов должна быть уникальной.
- `DISTINCT` можно использовать только для выборки данных, где вам нужно получить уникальные значения, например, для получения списка уникальных значений какого-либо столбца или комбинации столбцов.

Пример 4: Уникальные комбинации возраста и веса

Если мы хотим узнать все уникальные комбинации возраста и веса гусей, то можно использовать такой запрос:

```
SELECT DISTINCT age, weight
FROM geese;
```

Этот запрос вернет все уникальные сочетания возраста и веса, исключая дубли.



The screenshot shows a SQL IDE interface. The top pane contains the following SQL query:

```
1 SELECT DISTINCT egg_color
2 FROM eggs;
```

Below the query editor, there are tabs for "Data Output", "Messages", and "Notifications". The "Data Output" tab is active, displaying a table of results. The table has a header row with the column name "egg_color" and its data type "character varying (50)". There are three rows of data:

	egg_color character varying (50)
1	Серое
2	Темное
3	Белое



SELECT DISTINCT

12. Inner Join

Оператор INNER JOIN

`INNER JOIN` используется для соединения строк из двух или более таблиц на основе условия, которое должно быть выполнено для объединения данных. В запросе `INNER JOIN` обычно указывается, как сравниваются строки, например, по первичному ключу одной таблицы и внешнему ключу другой. Результатом запроса являются строки, удовлетворяющие условиям соединения, из всех таблиц, участвующих в `JOIN`.

Общий синтаксис INNER JOIN

```
SELECT столбцы
FROM таблица1
    [INNER] JOIN таблица2
    ON условие1
    [[INNER] JOIN таблица3
    ON условие2];
```

После оператора `JOIN` идет название таблицы, данные из которой нужно добавить в выборку. Ключевое слово `INNER` является необязательным, его отсутствие не влияет на выполнение запроса. Условие после `ON` определяет, каким образом происходит сравнение строк из разных таблиц.

Пример использования INNER JOIN с данными вашей базы

Для примера возьмем несколько таблиц из вашей базы данных: `geese`, `owners`, и `goose_owners`. Мы соединим таблицы, чтобы получить информацию о каждом гусе вместе с данными о владельце.

Запрос: Получение данных о гусе и его владельце

```
SELECT geese.name AS GooseName, geese.breed, owners.first_name AS
OwnerFirstName, owners.last_name AS OwnerLastName
FROM geese
JOIN goose_owners ON geese.goose_id = goose_owners.goose_id
JOIN owners ON owners.owner_id = goose_owners.owner_id;
```

Этот запрос возвращает информацию о каждом гусе, включая имя, породу, и данные о владельце (имя и фамилию). `JOIN` соединяет таблицы по полям `geese.goose_id` и `goose_owners.goose_id`, а также `owners.owner_id` и `goose_owners.owner_id`, чтобы найти соответствующего владельца для каждого гуся.

Использование псевдонимов таблиц для сокращения кода

Чтобы сократить код, можно использовать псевдонимы таблиц. Псевдонимы позволяют указывать короткие имена вместо полных названий таблиц.

Пример с псевдонимами

```
SELECT g.name AS GooseName, g.breed, o.first_name AS OwnerFirstName,
o.last_name AS OwnerLastName
FROM geese AS g
JOIN goose_owners AS go ON g.goose_id = go.goose_id
JOIN owners AS o ON o.owner_id = go.owner_id;
```

Соединение нескольких таблиц с фильтрацией

Допустим, нужно выбрать всех гусей, посещавших ветеринара с диагнозом "Здоров".

```
SELECT g.name AS GooseName, v.visit_date, v.diagnosis
FROM geese AS g
JOIN vet_visits AS v ON g.goose_id = v.goose_id
WHERE v.diagnosis = 'Здоров';
```

Пример соединения с более сложным условием

Предположим, мы хотим выбрать всех гусей, у которых есть владелец с фамилией "Иванов", и найти их вес.

```
SELECT g.name AS GooseName, g.weight, o.last_name AS OwnerLastName
FROM geese AS g
JOIN goose_owners AS go ON g.goose_id = go.goose_id
JOIN owners AS o ON o.owner_id = go.owner_id AND o.last_name = 'Иванов';
```

Здесь используется условие `ON`, чтобы соединить таблицы по `goose_id` и `owner_id`, а также фильтруются результаты только для тех владельцев, чья фамилия "Иванов".

Полезные рекомендации при использовании `INNER JOIN`

1. **Выбирайте только нужные столбцы:** Указывайте только те данные, которые нужны для запроса, чтобы избежать лишних вычислений.
2. **Фильтруйте данные до соединения:** Если возможно, добавьте условия `WHERE` до или после `JOIN`, чтобы уменьшить объем данных для обработки.
3. **Избегайте лишних соединений:** Чем больше таблиц в `JOIN`, тем больше ресурсов требуется для выполнения запроса.

The screenshot shows a SQL IDE interface. At the top, a query is entered in a text area:

```
1 SELECT g.name AS GooseName, g.weight, o.last_name AS OwnerLastName
2 FROM geese AS g
3 JOIN goose_owners AS go ON g.goose_id = go.goose_id
4 JOIN owners AS o ON o.owner_id = go.owner_id AND o.last_name = 'Иванов';
5
```

Below the query editor, there are tabs for "Data Output", "Messages", and "Notifications". The "Data Output" tab is active, showing a table of results. The table has three columns: "goosename", "weight", and "ownerlastname". The data is as follows:

	goosename character varying (100)	weight numeric (5,2)	ownerlastname character varying (100)
1	Гусь 1	5.20	Иванов
2	Гусь 4	7.30	Иванов

13. Left Join

Оператор `LEFT JOIN`

`LEFT JOIN` используется для объединения данных из двух таблиц. Он возвращает все строки из левой таблицы (указанной до `JOIN`), а также соответствующие строки из правой таблицы. Если в правой таблице нет совпадающей строки, `LEFT JOIN` вернет `NULL` для столбцов правой таблицы.

`LEFT JOIN` полезен для получения всех записей из одной таблицы с соответствующими данными из другой таблицы или для нахождения записей, которым не соответствует ни одна строка во второй таблице.

Синтаксис `LEFT JOIN`

```
SELECT столбцы
FROM таблица1
LEFT JOIN таблица2
ON условие_сопоставления;
```

Примеры использования `LEFT JOIN` с данными вашей базы данных

Предположим, у вас есть таблицы `geese`, `owners`, и `goose_owners`, которые связывают гусей с их владельцами.

Пример 1: Найти всех гусей и их владельцев (включая гусей без владельцев)

Этот запрос позволяет получить список всех гусей с информацией о владельцах. Гуси, у которых нет владельцев, будут возвращены с `NULL` в столбцах владельцев.

```
SELECT geese.name AS GooseName, owners.first_name AS OwnerFirstName,
owners.last_name AS OwnerLastName
FROM geese
LEFT JOIN goose_owners ON geese.goose_id = goose_owners.goose_id
LEFT JOIN owners ON goose_owners.owner_id = owners.owner_id;
```

Этот запрос возвращает всех гусей, а также имена и фамилии их владельцев, если такие владельцы существуют. Для гусей без владельцев столбцы `OwnerFirstName` и `OwnerLastName` будут `NULL`.

Пример 2: Найти всех владельцев и их гусей (включая владельцев без гусей)

Если вы хотите получить список всех владельцев, независимо от того, есть у них гуси или нет, можно использовать `LEFT JOIN`, начиная с таблицы `owners`.

```
SELECT owners.first_name AS OwnerFirstName, owners.last_name AS
OwnerLastName, geese.name AS GooseName
FROM owners
LEFT JOIN goose_owners ON owners.owner_id = goose_owners.owner_id
LEFT JOIN geese ON goose_owners.goose_id = geese.goose_id;
```

Этот запрос возвращает всех владельцев, а также имена гусей, если такие гуси есть. Если у владельца нет гусей, `GooseName` будет `NULL`.

Пример 3: Найти всех гусей и их последний визит к ветеринару (включая гусей без визитов)

Допустим, вы хотите узнать дату последнего визита к ветеринару для каждого гуся, включая тех, кто никогда не посещал ветеринара.

```
SELECT geese.name AS GooseName, MAX(vet_visits.visit_date) AS LastVisitDate
FROM geese
LEFT JOIN vet_visits ON geese.goose_id = vet_visits.goose_id
GROUP BY geese.name;
```

Этот запрос возвращает список всех гусей и дату их последнего визита к ветеринару. Если у гуся нет записей о посещениях, `LastVisitDate` будет `NULL`.

Пример 4: Найти гусей и корм, который они получают (включая гусей без назначенного корма)

Чтобы получить список всех гусей и их корм, включая гусей, для которых еще не указано расписание кормления, используем `LEFT JOIN` с таблицей `feeding_schedule`.

```
SELECT geese.name AS GooseName, feed.feed_name AS FeedName
FROM geese
LEFT JOIN feeding_schedule ON geese.goose_id = feeding_schedule.goose_id
LEFT JOIN feed ON feeding_schedule.feed_id = feed.feed_id;
```

Этот запрос возвращает всех гусей, с указанием корма, который они получают, если такой корм есть. Если у гуся нет назначенного корма, `FeedName` будет `NULL`.

Пример 5: Найти всех владельцев и тип корма, которым они кормят своих гусей (включая владельцев без кормления)

Этот запрос позволяет отобразить всех владельцев и информацию о корме, который получают их гуси. Если у владельца нет гусей или гуси не получают корм, столбец `FeedName` будет содержать `NULL`.

```
SELECT owners.first_name AS OwnerFirstName, owners.last_name AS
OwnerLastName, feed.feed_name AS FeedName
FROM owners
LEFT JOIN goose_owners ON owners.owner_id = goose_owners.owner_id
LEFT JOIN feeding_schedule ON goose_owners.goose_id =
```

```
feeding_schedule.goose_id  
LEFT JOIN feed ON feeding_schedule.feed_id = feed.feed_id;
```

Этот запрос позволяет увидеть всех владельцев и корм, который получают их гуси, если он существует. Если у владельца нет гусей или не указано кормление, столбец `FeedName` будет `NULL`.

14. Right Join

Оператор RIGHT JOIN

Оператор `RIGHT JOIN` (или `RIGHT OUTER JOIN`) используется для объединения данных из двух таблиц. Он возвращает все строки из правой таблицы (указанной после `JOIN`), а также соответствующие строки из левой таблицы. Если в левой таблице нет совпадающих строк, то для этих строк возвращаются `NULL` значения в столбцах левой таблицы.

`RIGHT JOIN` противоположен `LEFT JOIN`. Вместо того чтобы сохранять все строки из левой таблицы, он сохраняет все строки из правой таблицы, и если не находится совпадений в левой таблице, то эти строки будут содержать `NULL` для столбцов левой таблицы.

Синтаксис RIGHT JOIN

```
SELECT столбцы  
FROM таблица1  
RIGHT JOIN таблица2  
ON условие_сопоставления;
```

Примеры использования RIGHT JOIN с данными вашей базы данных

Предположим, у вас есть таблицы `geese`, `owners`, и `goose_owners`, которые связывают гусей с их владельцами. Рассмотрим несколько примеров использования `RIGHT JOIN`.

Пример 1: Найти всех владельцев и их гусей (включая владельцев без гусей)

Этот запрос позволяет получить список всех владельцев и гусей, которыми они владеют. Если у владельца нет гусей, то столбцы для гуся будут содержать `NULL`.

```
SELECT owners.first_name AS OwnerFirstName, owners.last_name AS  
OwnerLastName, geese.name AS GooseName  
FROM owners  
RIGHT JOIN goose_owners ON owners.id = goose_owners.owner_id  
RIGHT JOIN geese ON goose_owners.goose_id = geese.id;
```

Этот запрос возвращает всех владельцев и имена гусей, которыми они владеют. Если у владельца нет гусей, то столбец `GooseName` будет содержать `NULL`.

Пример 2: Найти всех гусей и их владельцев (включая гусей без владельцев)

Если вы хотите получить список всех гусей с их владельцами (включая гусей без владельцев), можно использовать `RIGHT JOIN` с таблицей `goose_owners` и `owners`.

```
SELECT geese.name AS GooseName, owners.first_name AS OwnerFirstName,
owners.last_name AS OwnerLastName
FROM geese
RIGHT JOIN goose_owners ON geese.id = goose_owners.goose_id
RIGHT JOIN owners ON goose_owners.owner_id = owners.id;
```

Этот запрос возвращает всех гусей, а также информацию о владельцах. Если гусь не имеет владельца, то столбцы владельцев будут содержать `NULL`.

Пример 3: Найти все кормления и связанные с ними гуся и их владельцы (включая кормления без гусей)

Чтобы найти все кормления и связанные с ними гуся и владельцев, включая кормления, которые не связаны с гусями, используем `RIGHT JOIN`.

```
SELECT feed.feed_name AS FeedName, geese.name AS GooseName,
owners.first_name AS OwnerFirstName
FROM feed
RIGHT JOIN feeding_schedule ON feed.id = feeding_schedule.feed_id
RIGHT JOIN geese ON feeding_schedule.goose_id = geese.id
RIGHT JOIN owners ON geese.id = owners.id;
```

Этот запрос возвращает все кормления, имена гусей и владельцев. Если кормление не связано с гусями, то столбцы для гуся и владельца будут содержать `NULL`.

Пример 4: Найти всех ветеринаров и их визиты, включая визиты без ветеринаров

Этот запрос позволяет получить все записи о визитах к ветеринарам, включая те, у которых нет информации о ветеринаре.

```
SELECT vet_visits.visit_date AS VisitDate, geese.name AS GooseName,
vet_visits.diagnosis AS Diagnosis
FROM vet_visits
RIGHT JOIN geese ON vet_visits.goose_id = geese.id;
```


Этот запрос вернет все визиты к ветеринару, а также имена гусей и диагнозы. Если визит не связан с гусем, столбец `GooseName` будет содержать `NULL`.

Пример 5: Найти всех гусей и корм, который они получают (включая гусей без корма)

Если необходимо найти всех гусей и их корм, но включать тех, кто не имеет кормления, то используем `RIGHT JOIN` с таблицами `geese`, `feeding_schedule` и `feed`.

```
SELECT geese.name AS GooseName, feed.feed_name AS FeedName
FROM geese
RIGHT JOIN feeding_schedule ON geese.id = feeding_schedule.goose_id
RIGHT JOIN feed ON feeding_schedule.feed_id = feed.id;
```

Этот запрос возвращает всех гусей и корм, который они получают, даже если у гуся нет назначения на корм (в таком случае столбец `FeedName` будет содержать `NULL`).

Основные различия между `LEFT JOIN` и `RIGHT JOIN`

- `LEFT JOIN` возвращает все строки из левой таблицы и соответствующие строки из правой таблицы. Если строки правой таблицы не существует, будут возвращены `NULL` для столбцов правой таблицы.
- `RIGHT JOIN` возвращает все строки из правой таблицы и соответствующие строки из левой таблицы. Если строки левой таблицы не существует, будут возвращены `NULL` для столбцов левой таблицы.

Оба оператора полезны для извлечения данных, когда необходимо сохранить все строки из одной из таблиц, независимо от того, есть ли для них соответствующие строки в другой таблице.

15. Cross Join

Оператор `CROSS JOIN` используется для выполнения декартова произведения двух таблиц, при котором каждая строка из первой таблицы соединяется с каждой строкой из второй таблицы, формируя все возможные комбинации. Это может быть полезно для создания пар между элементами двух таблиц, когда не нужно учитывать совпадения по какому-либо критерию.

Примеры использования `CROSS JOIN` в вашей базе данных

Пример 1: Декартово произведение всех гусей и кормов

Этот запрос создает все возможные комбинации гусей и кормов, которые могут быть предложены каждому гусю.

```
SELECT geese.name AS GooseName, feed.feed_name AS FeedName
FROM geese
CROSS JOIN feed;
```

Этот запрос возвращает строки, представляющие сочетания каждого гуся с каждым кормом. Если в `geese` 10 строк, а в `feed` — 5 строк, итоговый результат будет содержать 50 строк ($10 * 5$).

Пример 2: Декартово произведение всех владельцев и кормов

В данном примере генерируются все возможные пары владельцев и кормов.

```
SELECT owners.first_name AS OwnerFirstName, feed.feed_name AS FeedName
FROM owners
CROSS JOIN feed;
```

Каждое имя владельца будет связано с каждым видом корма. Такой запрос может быть полезен для анализа предпочтений или назначений кормов владельцам.

Пример 3: Декартово произведение гусей, владельцев и кормов

Здесь создается список всех возможных сочетаний гусей, их владельцев и кормов.

```
SELECT geese.name AS GooseName, owners.first_name AS OwnerFirstName,
feed.feed_name AS FeedName
FROM geese
CROSS JOIN owners
CROSS JOIN feed;
```

Результатом будет все возможные комбинации, что может быть полезно для анализа соответствий всех владельцев и всех кормов с гусями.

Важные аспекты использования `CROSS JOIN`

1. **Объем данных:** Результат `CROSS JOIN` растет в геометрической прогрессии, пропорционально количеству строк в каждой таблице. Например, если в одной таблице 1000 строк, а в другой 5000, результат будет содержать 5 миллионов строк. Поэтому его следует применять осторожно.
2. **Типы задач:** `CROSS JOIN` полезен для генерации всех возможных комбинаций элементов из двух таблиц, таких как сочетания продуктов и клиентов, для анализа или

моделирования.

3. **Отсутствие условий соединения:** В отличие от `INNER JOIN` и `LEFT JOIN`, в `CROSS JOIN` нет условий объединения — всегда генерируются все возможные сочетания.

Примечание

- `CROSS JOIN` наиболее полезен в случаях, когда нужно построить комбинации данных, а не только соответствия по ключам.
- Результаты такого запроса могут быть огромными, поэтому важно учитывать это при работе с большими таблицами и использовать `LIMIT`, чтобы уменьшить объем данных.

```
1 SELECT geese.name AS GooseName, owners.first_name AS OwnerFirstName, feed.feed_name AS FeedName
2 FROM geese
3 CROSS JOIN owners
4 CROSS JOIN feed;
```

	goosename character varying (100)	ownerfirstname character varying (100)	feedname character varying (100)
1	Гусь 1	Иван	Корм 1
2	Гусь 1	Иван	Корм 2
3	Гусь 1	Иван	Корм 3
4	Гусь 1	Петр	Корм 1
5	Гусь 1	Петр	Корм 2
6	Гусь 1	Петр	Корм 3
7	Гусь 1	Сергей	Корм 1
8	Гусь 1	Сергей	Корм 2
9	Гусь 1	Сергей	Корм 3
10	Гусь 2	Иван	Корм 1
11	Гусь 2	Иван	Корм 2
12	Гусь 2	Иван	Корм 3
13	Гусь 2	Петр	Корм 1
14	Гусь 2	Петр	Корм 2
15	Гусь 2	Петр	Корм 3
16	Гусь 2	Сергей	Корм 1
17	Гусь 2	Сергей	Корм 2
18	Гусь 2	Сергей	Корм 3
19	Гусь 3	Иван	Корм 1
20	Гусь 3	Иван	Корм 2
21	Гусь 3	Иван	Корм 3
22	Гусь 3	Петр	Корм 1

Функция `AVG` возвращает среднее значение для набора числовых значений. Эта функция полезна для вычисления среднего показателя, например, для анализа цен или рейтингов.

Синтаксис:

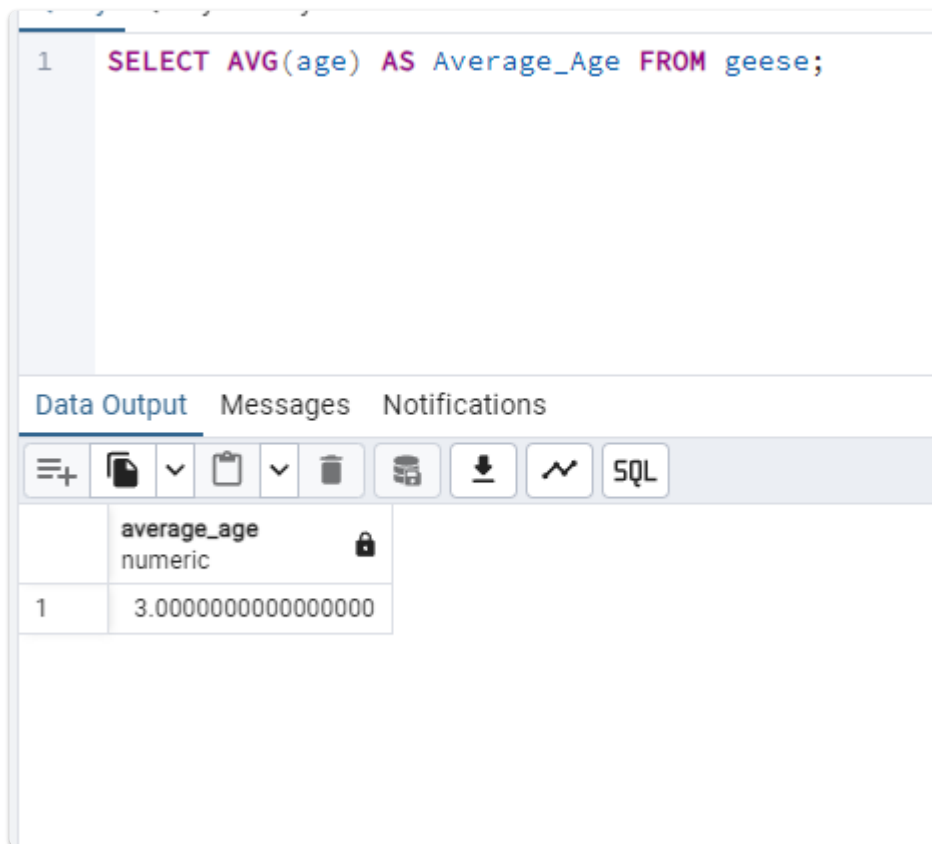
```
SELECT AVG(столбец) FROM таблица;
```

Пример:

Найдем средний возраст гусей в таблице `geese`:

```
SELECT AVG(age) AS Average_Age FROM geese;
```

Этот запрос вернет средний возраст всех записей в столбце `age` таблицы `geese`. Если есть значения `NULL`, они игнорируются.



The screenshot shows a SQL query editor with the following query:

```
1 SELECT AVG(age) AS Average_Age FROM geese;
```

Below the query editor, there is a pane with three tabs: "Data Output", "Messages", and "Notifications". The "Data Output" tab is active, showing a table with the results of the query.

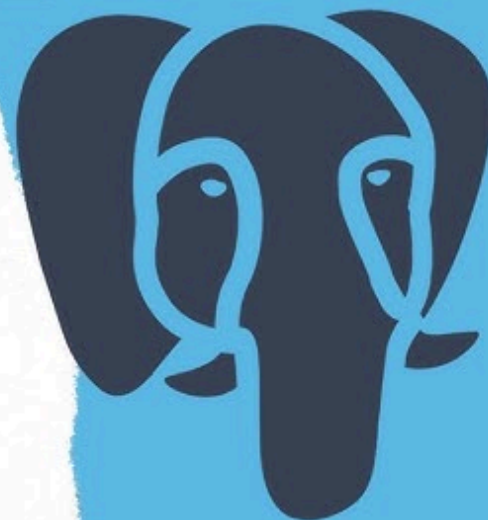
	average_age numeric	
1	3.0000000000000000	

POSTGRESQL

avg()

function

avg()- calculate
average value



17. SUM

Функция `SUM` возвращает сумму значений, что особенно полезно для расчета итогов по числовым данным. Например, она помогает определить суммарные продажи, общий вес или стоимость товаров.

Синтаксис:

```
SELECT SUM(столбец) FROM таблица;
```

Пример:

Подсчитаем общий вес всех гусей в таблице `geese`:

```
SELECT SUM(weight) AS Total_Weight FROM geese;
```

Здесь функция `SUM` возвращает суммарное значение по всем значениям в столбце `weight`. Как и `AVG`, эта функция игнорирует значения `NULL`.

```
1 SELECT SUM(weight) AS Total_Weight FROM geese;
```

Data Output Messages Notifications

	total_weight numeric
1	27.00



18. COUNT

Функция `COUNT` возвращает количество строк в наборе данных. Она бывает полезна для подсчета количества записей, подходящих под определенные условия, например, для подсчета количества клиентов или заказов.

Синтаксис:

```
SELECT COUNT(*) FROM таблица;  
SELECT COUNT(столбец) FROM таблица;
```

Пример:

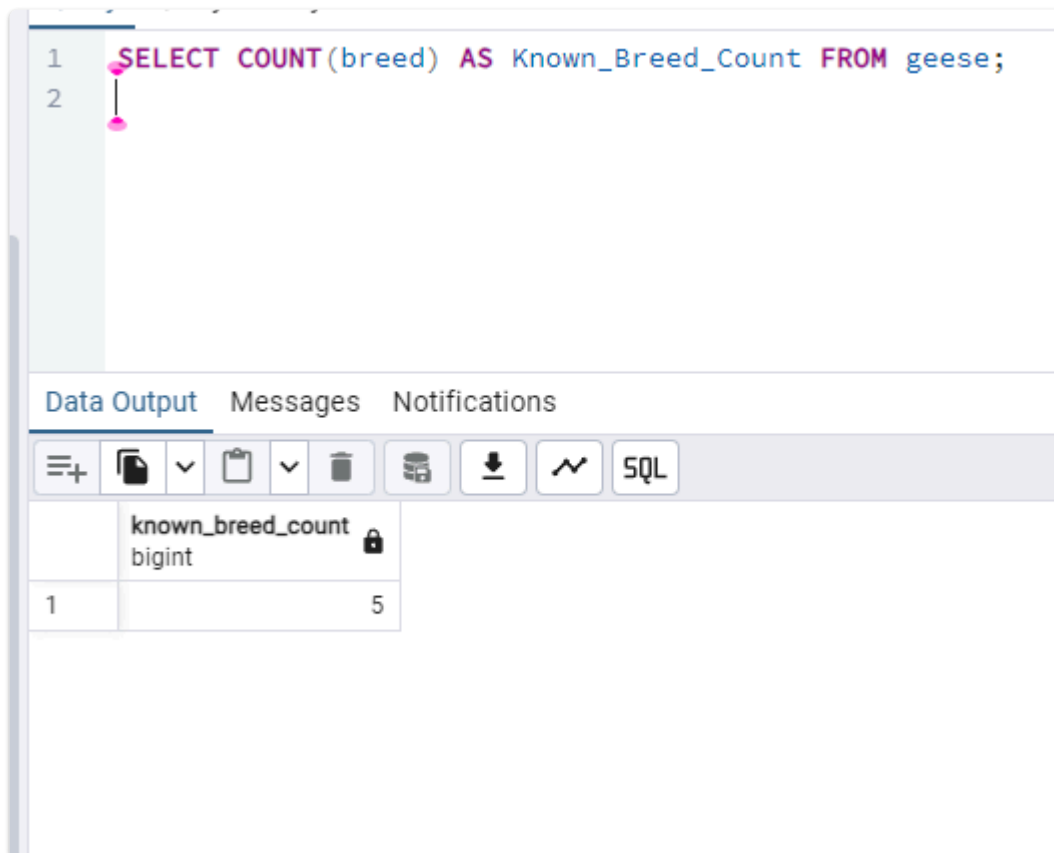
Подсчитаем количество гусей в таблице `geese`:

```
SELECT COUNT(*) AS Total_Geese FROM geese;
```

Этот запрос вернет общее число строк в таблице. Если указать столбец в `COUNT`, как в `COUNT(столбец)`, то `NULL`-значения будут игнорироваться. Например, подсчитаем количество гусей с известной породой:

```
SELECT COUNT(breed) AS Known_Breed_Count FROM geese;
```

Здесь `COUNT(breed)` не учитывает строки с `NULL` в столбце `breed`.



The screenshot shows a SQL IDE interface. The top pane contains the following SQL query:

```
1 SELECT COUNT(breed) AS Known_Breed_Count FROM geese;  
2
```

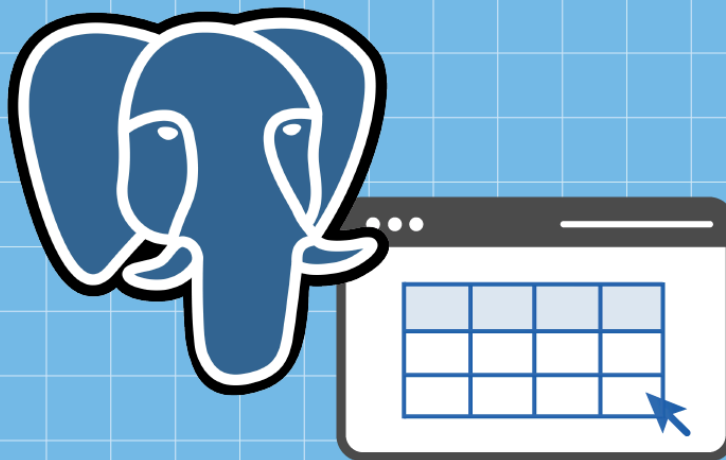
The bottom pane is titled "Data Output" and shows the results of the query. It includes a toolbar with icons for file operations, a table view icon, a download icon, a refresh icon, and an "SQL" button. The results are displayed in a table with the following structure:

	known_breed_count	
	bigint	
1	5	

```
1 SELECT COUNT(*) AS Total_Geese FROM geese;
```

Data Output Messages Notifications

	total_geese bigint	
1		5



19. Подзапросы

Подзапросы — это операторы `SELECT`, встроенные в другие запросы SQL, что позволяет выполнять дополнительные операции и возвращать результаты, которые можно использовать в основном запросе. Подзапросы могут находиться в различных частях основного запроса: в `WHERE`, `FROM`, `SELECT`, и других частях. Они полезны для решения задач, которые сложно решить одним запросом, или для получения данных, используемых в основном запросе.

Простейший пример подзапроса

Допустим, у нас есть таблица `geese`, и мы хотим выбрать всех гусей, чей вес превышает средний вес всех гусей. В этом случае можно использовать подзапрос, который сначала вычислит средний вес, а затем передаст его в основной запрос для фильтрации.

```
SELECT name, weight
FROM geese
WHERE weight > (SELECT AVG(weight) FROM geese);
```

В этом примере подзапрос `(SELECT AVG(weight) FROM geese)` возвращает средний вес всех гусей, а основной запрос выбирает только тех гусей, у которых вес больше этого значения.

Подзапросы в других частях запроса

Подзапросы можно использовать и в других частях основного запроса, например, в `FROM`, чтобы создать временные таблицы, или в `SELECT`, чтобы добавить дополнительные вычисляемые столбцы.

Пример подзапроса в `FROM`

Если мы хотим узнать породы и количество гусей в каждой породе, чей вес выше среднего, можно воспользоваться подзапросом в `FROM`:

```
SELECT breed, COUNT(*) AS AboveAverageWeightCount
FROM geese
WHERE weight > (SELECT AVG(weight) FROM geese)
GROUP BY breed;
```

Здесь подзапрос выполняет вычисление среднего веса, а основной запрос фильтрует гусей, у которых вес выше среднего, и группирует их по породе.

Коррелированный подзапрос

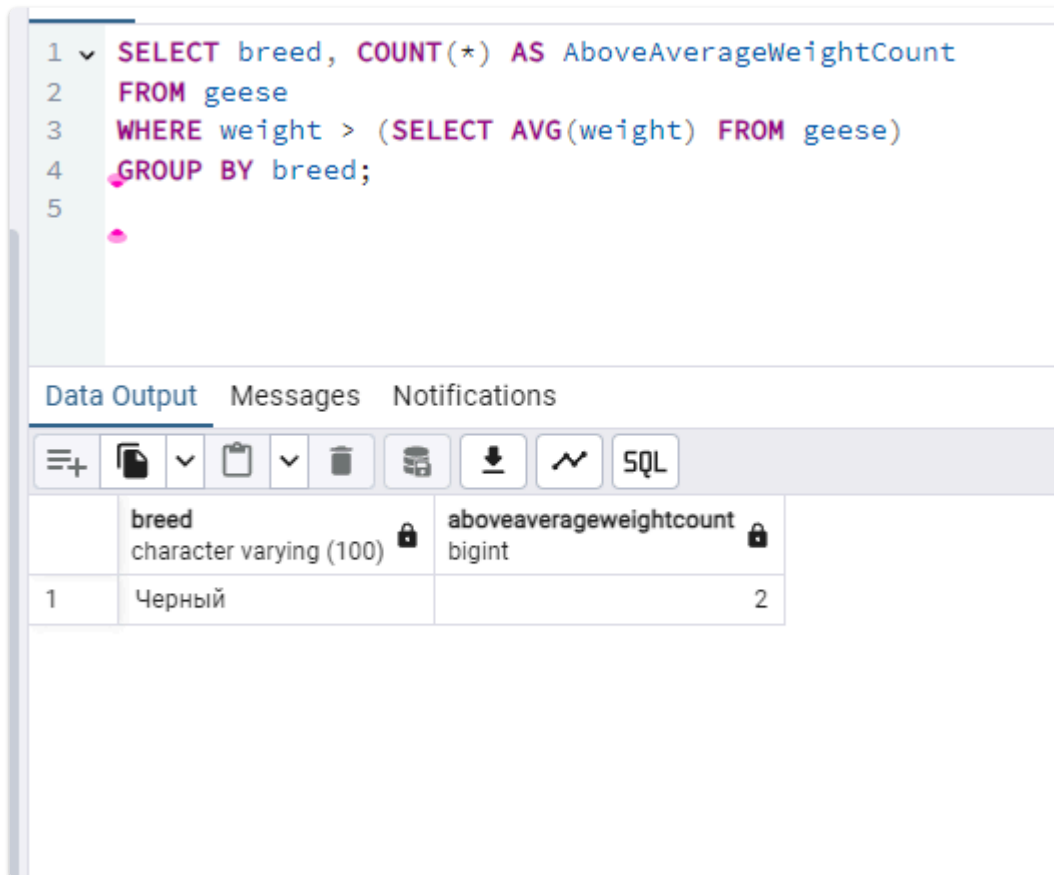
Иногда подзапрос может зависеть от строки основного запроса. Такой подзапрос называется коррелированным, так как он выполняется отдельно для каждой строки основного запроса.

Пример коррелированного подзапроса

Например, если мы хотим получить гусей, которые имеют больший вес, чем средний вес для их породы, можно использовать коррелированный подзапрос:

```
SELECT name, breed, weight
FROM geese g1
WHERE weight > (SELECT AVG(weight) FROM geese g2 WHERE g2.breed = g1.breed);
```

В этом примере подзапрос рассчитывает средний вес для каждой породы гусей отдельно, и основной запрос выбирает гусей, которые превышают этот средний вес.



The screenshot shows a SQL IDE interface. The top pane contains the following SQL query:

```
1 SELECT breed, COUNT(*) AS AboveAverageWeightCount
2 FROM geese
3 WHERE weight > (SELECT AVG(weight) FROM geese)
4 GROUP BY breed;
5
```

The bottom pane is titled "Data Output" and shows the results of the query. It includes a toolbar with icons for file operations and a "SQL" button. The results are displayed in a table with two columns: "breed" (character varying (100)) and "aboveaverageweightcount" (bigint). The table contains one row with the value "Черный" in the "breed" column and "2" in the "aboveaverageweightcount" column.

	breed character varying (100)	aboveaverageweightcount bigint
1	Черный	2

20. Конкатенация столбцов

В PostgreSQL конкатенация строк также возможна с использованием функции `CONCAT()` или оператора `||` (двойной вертикальной черты). Рассмотрим оба варианта:

1. Использование функции `CONCAT()`

В PostgreSQL функция `CONCAT()` работает аналогично MySQL и соединяет два или более строковых значения в одну строку.

Пример использования `CONCAT()`:

```
SELECT CONCAT(geese.name, ' weighs ', geese.weight, ' kg') AS GooseInfo
FROM geese;
```

Этот запрос создаёт строку с информацией о гусях, например, "Goose1 weighs 5 kg".

2. Использование оператора `||` (два знака вертикальной черты)

В PostgreSQL для конкатенации строк также можно использовать оператор `||`, который выполняет ту же задачу.

Пример использования оператора `||`:

```
SELECT geese.name || ' weighs ' || geese.weight || ' kg' AS GooseInfo
FROM geese;
```

Этот запрос также создаёт строку с информацией о гусях, но с использованием оператора `||` для объединения столбцов.

Примеры использования конкатенации в PostgreSQL с данными вашей базы данных

1. Конкатенация имени и фамилии владельцев гусей

Предположим, что в вашей базе данных есть таблица `owners`, содержащая столбцы `first_name` и `last_name`. Вы хотите объединить имя и фамилию владельцев в один столбец.

```
SELECT CONCAT(owners.first_name, ' ', owners.last_name) AS FullName
FROM owners;
```

Используя оператор `||`, это будет выглядеть так:

```
SELECT owners.first_name || ' ' || owners.last_name AS FullName
FROM owners;
```

2. Конкатенация имени и веса гусей

```
SELECT CONCAT(geese.name, ' weighs ', geese.weight, ' kg') AS GooseInfo
FROM geese;
```

Используя оператор `||`:

```
SELECT geese.name || ' weighs ' || geese.weight || ' kg' AS GooseInfo
FROM geese;
```

3. Конкатенация нескольких столбцов в таблице `Orders`

```
SELECT CONCAT(Products.ProductName, ' (' , Orders.ProductCount, ' units) on
', Orders.CreatedAt) AS OrderDetails
FROM Orders
JOIN Products ON Products.Id = Orders.ProductId;
```

Используя оператор `||`:

```
SELECT Products.ProductName || ' (' || Orders.ProductCount || ' units) on '
|| Orders.CreatedAt AS OrderDetails
FROM Orders
JOIN Products ON Products.Id = Orders.ProductId;
```

4. Добавление дополнительного текста и конкатенация с датой

```
SELECT CONCAT('Goose ', geese.name, ' was born on ',
TO_CHAR(geese.date_of_birth, 'YYYY-MM-DD')) AS BirthDetails
FROM geese;
```

Используя оператор `||`:

```
SELECT 'Goose ' || geese.name || ' was born on ' ||
TO_CHAR(geese.date_of_birth, 'YYYY-MM-DD') AS BirthDetails
FROM geese;
```

Важные моменты в PostgreSQL:

- **Использование `CONCAT()`**: Это функция, которая объединяет строки, и в отличие от оператора `||`, она автоматически обрабатывает `NULL` значения. Если один из аргументов `NULL`, то функция `CONCAT()` просто пропустит его, а не вернёт `NULL`. Например:

```
SELECT CONCAT(goose.name, ' ', goose.weight) AS GooseInfo
FROM goose;
```

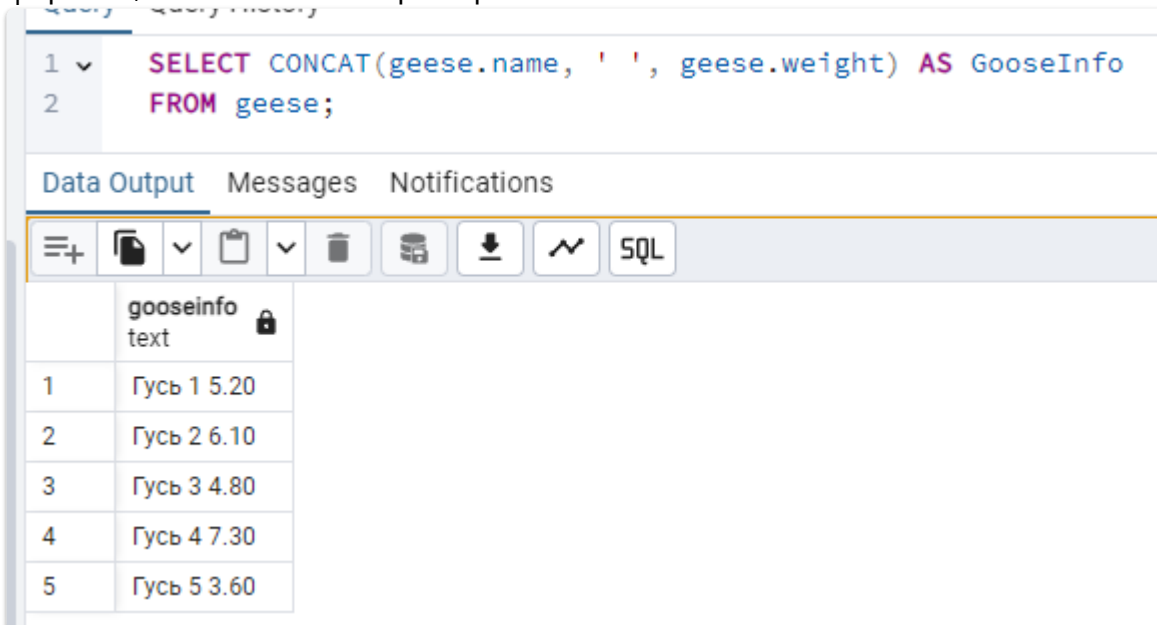
Если в каком-то столбце значение `NULL`, то `CONCAT()` всё равно вернёт корректную строку без ошибок.

- **Использование оператора `||`**: Оператор конкатенации `||` в PostgreSQL требует, чтобы оба операнда были строками. Если один из операндов — это `NULL`, то результат будет `NULL`. Чтобы избежать этого, можно использовать функцию `COALESCE()`:

```
SELECT goose.name || ' weighs ' || COALESCE(goose.weight::TEXT,
'Unknown') || ' kg' AS GooseInfo
FROM goose;
```

В этом примере, если значение `goose.weight` равно `NULL`, то оно будет заменено на `'Unknown'`.

- **Форматирование дат**: В PostgreSQL для форматирования даты используется функция `TO_CHAR()`. Это позволяет конкатенировать даты с другими строками в нужном формате, как показано в примере выше.



The screenshot shows a PostgreSQL client interface. At the top, a SQL query is entered in a text area:

```
1 SELECT CONCAT(goose.name, ' ', goose.weight) AS GooseInfo
2 FROM goose;
```

Below the query area, there are tabs for "Data Output", "Messages", and "Notifications". The "Data Output" tab is active, showing a table of results. The table has a single column named "gooseinfo" of type "text". The results are as follows:

	gooseinfo
1	Гусь 1 5.20
2	Гусь 2 6.10
3	Гусь 3 4.80
4	Гусь 4 7.30
5	Гусь 5 3.60