

Microsoft Copilot: ваш ИИ-помощник

 copilot.microsoft.com/chats/mpKkpH2UueJVNRugQepF5



Сегодня

`du -sh /var/lib/pgsql/14/messenger/base/16709033/23835271*` необходимо подсчитать суммарный реальный размер на диске в байтах

Чтобы определить суммарный реальный размер файлов на диске в байтах, можно использовать команду:

`bash`

```
du -cb /var/lib/pgsql/14/messenger/base/16709033/23835271*
```

Флаг `-c` суммирует общий размер, а `-b` отображает размер в байтах. В конце вывода будет указана строка `total`, содержащая общий объем занимаемого пространства. Попробуйте и скажите, получилось ли у вас! 🚀

18619056128 байт в Гб , три знака после запятой

Чтобы перевести байты в гигабайты с тремя знаками после запятой, нужно разделить на 1024^3 :

$$18619056128 \div 1073741824 \approx 17.338 \text{ ГБ}$$
$$18619056128 \text{ \div 1073741824 \approx 17.338 \text{ ГБ}}$$

Таким образом, размер составляет **17.338 ГБ**. 🚀

Postgresql тип данных boolean сколько байт занимает с учетом выравнивания

В PostgreSQL тип данных `boolean` занимает **1 байт**. Однако, с учетом выравнивания, его фактический размер в строке таблицы может увеличиваться.

На 64-битных системах PostgreSQL использует **выравнивание на 8 байт** для данных в строке. Это означает, что если `boolean` стоит после типа данных с более строгими требованиями к выравниванию, может добавляться **заполняющий байт**. Например, если `boolean` идет после `int2` (который требует 2-байтового выравнивания), то общий размер строки увеличится на 3 байта: 1 байт для `boolean` и 2 байта для `int2`.

Если `boolean` используется в одиночной колонке, его размер будет **1 байт**, но в зависимости от расположения в строке таблицы и соседних типов данных, фактический размер может быть больше из-за выравнивания.

Если вам нужно точно определить размер, можно использовать функцию:

sql

```
SELECT pg_column_size(row(true));
```

Это покажет реальный размер хранения `boolean` в контексте строки. 🚀

```
SELECT pg_column_size(row(true)); для какой таблицы?
```

Функция `pg_column_size(row(true))` не требует конкретной таблицы, так как `row(true)` создаёт временный кортеж и измеряет его размер. Это позволяет узнать, сколько байт занимает `boolean` в контексте строки без необходимости обращаться к реальным таблицам.

Если же вы хотите определить размер `boolean` в существующей таблице, можно использовать:

sql

```
SELECT pg_column_size(my_boolean_column) FROM my_table LIMIT 1;
```

Где:

- `my_boolean_column` — название столбца типа `boolean`.
- `my_table` — имя таблицы.

Так вы получите реальный размер хранения `boolean` в контексте строки вашей таблицы. 🚀

```
SELECT pg_column_size(row(true)); = 25 SELECT pg_column_size(click_text) FROM
messages LIMIT 1; = 1
```

Интересные результаты! 🤔

Функция `pg_column_size(row(true)) = 25` показывает, что при упаковке `boolean` в кортеж (`row()`) он занимает больше места из-за служебной информации о строке. Стандартная строка в PostgreSQL содержит заголовок (обычно 24 байта на 64-битных системах), плюс 1 байт на сам `boolean`, вот и выходит 25 байт.

А вот `SELECT pg_column_size(click_text) FROM messages LIMIT 1; = 1` говорит о том, что в вашей таблице `messages` поле `click_text` занимает 1 байт, что может означать, что оно хранит либо `NULL`, либо очень короткое значение.

Если интересно детальнее разобраться, можно проверить:

sql

```
SELECT pg_column_size(row(click_text)) FROM messages LIMIT 1;
```

Это покажет, сколько байтов `click_text` занимает в контексте строки, аналогично `row(true)`. 🚀

Таблица CREATE TABLE IF NOT EXISTS public.messages (id bigint NOT NULL DEFAULT nextval('messages_id_seq'::regclass), profile_id bigint NOT NULL, template_id bigint NOT NULL, send_status boolean, delivery_status boolean, close_status boolean, click_status boolean, read_status boolean, remove_status boolean, transaction_time bigint NOT NULL DEFAULT 0, click_text boolean DEFAULT false, click_button2 boolean DEFAULT false, click_button1 boolean DEFAULT false, click_button3 boolean DEFAULT false, attributes jsonb DEFAULT '{}':jsonb, time_to_send bigint, created_at bigint DEFAULT 0, trigger_uuid text COLLATE pg_catalog."default" NOT NULL DEFAULT '':text, try_integration_send boolean, repeat integer NOT NULL DEFAULT 0, CONSTRAINT messages_pkey PRIMARY KEY (profile_id, template_id, trigger_uuid, repeat)) WITH (FILLFACTOR = 90, autovacuum_analyze_scale_factor = 0.005, autovacuum_analyze_threshold = 50, autovacuum_vacuum_scale_factor = 0.001) TABLESPACE pg_default; каков расчетный размер одного кортежа?

Чтобы рассчитать размер одного кортежа в PostgreSQL, нужно учесть следующие факторы:

1. **Заголовок строки:** На 64-битных системах занимает **24 байта**.
2. **Размер данных:** Сумма размеров всех колонок, с учетом выравнивания.
3. **Выравнивание:** PostgreSQL выравнивает данные для эффективного хранения.
4. **TOAST:** Поля типа `jsonb` и `text` могут храниться отдельно.

Оценка размеров столбцов:

Тип данных	Столбец(-цы)	Размер (байт)
<code>bigint</code>	<code>id, profile_id, template_id, transaction_time, time_to_send, created_at, repeat</code>	$8 \times 7 = 56$
<code>boolean</code>	8 столбцов <code>boolean</code>	$1 \times 8 = 8$

Тип данных	Столбец(-цы)	Размер (байт)
<code>jsonb</code>	<code>attributes</code>	Зависит от данных (минимум 1 байт)
<code>text</code>	<code>trigger_uuid</code>	Зависит от данных (минимум 4 байта)
<code>integer</code>	<code>repeat</code>	4 байта
Итого (без выравнивания)		≈ 93+ байта

Выравнивание:

- `bigint` требует **8-байтового** выравнивания.
- `boolean` может занимать больше места из-за выравнивания.
- `jsonb` и `text` могут быть вынесены в TOAST, снижая размер.

Итоговая оценка:

- Базовый размер **≈ 93 байта + выравнивание**.
- Минимально **около 100-120 байт** на строку (без TOAST).
- Если `jsonb` и `text` хранятся отдельно, основной кортеж может быть **до 150 байт**.

Для точного измерения можно выполнить:

sql

```
SELECT pg_column_size(row(*)) FROM messages LIMIT 1;
```

Это покажет **реальный** размер кортежа с учетом выравнивания и хранимых данных. 🚀

```
[42601] ERROR: syntax error at or near "(*"
```

Ах, небольшая ошибка! В PostgreSQL нельзя использовать `row(*)`. Вместо этого попробуйте:

sql

```
SELECTpg_column_size(row(id, profile_id, template_id, send_status, delivery_status,
close_status, click_status, read_status, remove_status, transaction_time, click_text,
click_button2, click_button1, click_button3, attributes, time_to_send, created_at,
trigger_uuid, try_integration_send, repeat))
FROMmessages LIMIT 1;
```

Здесь я вручную перечислил все столбцы, чтобы создать корректный `row()`. Это должно дать точный размер кортежа. 🚀 Если PostgreSQL все равно ругается, попробуйте просто:

sql

```
SELECTpg_column_size(messages.*) FROMmessages LIMIT 1;
```

Этот вариант измеряет размер всей строки в `messages`. Попробуйте и расскажите, как получилось! 🔍

104

Отлично! Это значит, что реальный размер одного кортежа в вашей таблице `messages` составляет **104 байта**. 🎯

Этот результат включает все столбцы, заголовок строки и выравнивание. Возможно, значения `jsonb` и `text` (например, `attributes` и `trigger_uuid`) не увеличивают размер строки, так как они хранятся в TOAST. Если бы эти поля содержали длинные значения, размер кортежа мог бы быть больше.

Если хотите проверить средний размер строк в таблице, можно использовать:

sql

```
SELECTpg_relation_size('messages') /pg_table_size('messages') ASavg_tuple_size;
```

Это даст **средний размер** строк в вашей таблице, включая возможные TOAST-данные. 🚀 Как вам такие расчёты? 😊

```
SELECT pg_relation_size('messages') / pg_table_size('messages') AS avg_tuple_size; 0
```

Значение `0` означает, что `pg_relation_size('messages')` равно `0` или гораздо меньше `pg_table_size('messages')`, поэтому результат округляется до нуля.

Попробуйте вместо этого:

sql

```
SELECTpg_relation_size('messages')::numeric/pg_table_size('messages')::numericASavg_tuple_size;
```

Здесь я привожу значения к `numeric`, чтобы избежать округления до `0`. Это должно дать более точный результат. 🚀 Попробуйте и скажите, что получилось! 😊

```
0.99972193318036725059
```

Отлично, теперь мы видим, что средний размер кортежа в вашей таблице `messages` почти полностью соответствует размеру реальных данных! Значение ≈ 1 говорит о том, что большинство данных хранятся непосредственно в основном отношении таблицы, а не вынесены в TOAST.

Если `jsonb` или `text`-поля содержат длинные значения, они могут быть вынесены в TOAST, и тогда средний размер кортежа мог бы быть **меньше 1**. В вашем случае всё компактно хранится, что может быть хорошо для быстрого доступа.

Хотите проверить, есть ли TOAST-записи для `messages`? Можно выполнить:

sql

```
SELECTpg_relation_size('messages') AS table_size,  
       pg_total_relation_size('messages') AS total_size;
```

Это покажет разницу между основным размером таблицы и её полным размером (включая TOAST и индексы). 🚀 Любопытно, что получится! 😊

```
table_size;total_size 18613886976;35679387648
```

Интересные цифры! 📊

Ваши данные показывают, что:

- **Размер основной таблицы** (`table_size`): ~18.6 ГБ
- **Общий размер** (`total_size`): ~35.7 ГБ

Разница (~17.1 ГБ) говорит о том, что значительная часть данных **вынесена в TOAST или занята индексами**. Это может означать:

1. Поля `jsonb` (`attributes`) и `text` (`trigger_uuid`) хранятся в TOAST.
2. Индексы занимают значительное пространство.

Если хотите детально изучить TOAST, попробуйте:

sql

```
SELECT relname, pg_relation_size(oid) AS size
FROM pg_class
WHERE relname LIKE 'pg_toast_%';
```

Это покажет размер TOAST-таблиц, связанных с хранимыми длинными полями. 🚀 Хотите также посмотреть, сколько места занимают индексы? 😊

```
relname;size pg_toast_19499610;0 pg_toast_16709037;0 pg_toast_16709037_index;8192
pg_toast_19499610_index;8192 pg_toast_16709046;0 pg_toast_16709046_index;8192
pg_toast_16709055;0 pg_toast_16709055_index;8192 pg_toast_17017233;0
pg_toast_16709060;0 pg_toast_16709060_index;8192 pg_toast_17017233_index;8192
pg_toast_19499630;0 pg_toast_19499630_index;8192 pg_toast_19499665;0
pg_toast_19499665_index;8192 pg_toast_1255;8192 pg_toast_1255_index;16384
pg_toast_1247;0 pg_toast_1247_index;8192 pg_toast_2604;0 pg_toast_2604_index;8192
pg_toast_2606;0 pg_toast_2606_index;8192 pg_toast_2612;0 pg_toast_2612_index;8192
pg_toast_2600;0 pg_toast_2600_index;8192 pg_toast_2619;458752
pg_toast_2619_index;49152 pg_toast_3381;0 pg_toast_3381_index;8192 pg_toast_3429;0
pg_toast_3429_index;8192 pg_toast_2618;540672 pg_toast_2618_index;16384
pg_toast_2620;0 pg_toast_2620_index;8192 pg_toast_3466;0 pg_toast_3466_index;8192
pg_toast_2609;0 pg_toast_2609_index;8192 pg_toast_2615;0 pg_toast_2615_index;8192
pg_toast_1262;0 pg_toast_1262_index;8192 pg_toast_2964;0 pg_toast_2964_index;8192
pg_toast_1213;0 pg_toast_1213_index;8192 pg_toast_1260;0 pg_toast_1260_index;8192
pg_toast_2396;0 pg_toast_2396_index;8192 pg_toast_3600;0 pg_toast_3600_index;8192
pg_toast_3079;0 pg_toast_3079_index;8192 pg_toast_2328;0 pg_toast_2328_index;8192
pg_toast_1417;0 pg_toast_1417_index;8192 pg_toast_1418;0 pg_toast_1418_index;8192
pg_toast_3118;0 pg_toast_3118_index;8192 pg_toast_3256;0 pg_toast_3256_index;8192
pg_toast_6000;0 pg_toast_6000_index;8192 pg_toast_826;0 pg_toast_826_index;8192
pg_toast_3394;0 pg_toast_3394_index;8192 pg_toast_3596;0 pg_toast_3596_index;8192
pg_toast_3592;0 pg_toast_3592_index;8192 pg_toast_3456;0 pg_toast_3456_index;8192
pg_toast_3350;0 pg_toast_3350_index;8192 pg_toast_6100;0 pg_toast_6100_index;8192
pg_toast_19499693;0 pg_toast_17017259;0 pg_toast_19499693_index;8192
pg_toast_17017259_index;8192 pg_toast_17017297_index;8192
pg_toast_17017240_index;8192 pg_toast_17017297;0 pg_toast_17017271;0
pg_toast_17017271_index;8192 pg_toast_17017240;0 pg_toast_20860887;0
pg_toast_20860887_index;8192 pg_toast_17017291;0 pg_toast_17017291_index;8192
pg_toast_17017333_index;16384 pg_toast_17017333;65536 pg_toast_14172;0
pg_toast_14172_index;8192 pg_toast_14167;0 pg_toast_14167_index;8192 pg_toast_14177;0
pg_toast_14177_index;8192 pg_toast_14182;0 pg_toast_14182_index;8192
pg_toast_17017342;0 pg_toast_17017342_index;8192 pg_toast_17017357;0
pg_toast_17017357_index;8192 pg_toast_17017363;0 pg_toast_17017363_index;8192
pg_toast_17017392;0 pg_toast_17017392_index;8192 pg_toast_17017405;0
pg_toast_17017405_index;8192 pg_toast_17017426;0 pg_toast_17017426_index;8192
pg_toast_17017454_index;49152 pg_toast_17017454;827392 pg_toast_17017437;0
pg_toast_17017437_index;8192
```

Ваши TOAST-таблицы, связанные с таблицей `messages`, практически не занимают место — большинство из них имеют размер **0 байт**, а их индексы — по **8192 байта** (8 КБ). Это говорит о том, что длинные значения (например, `jsonb` и `text`) **не вынесены** в TOAST, а хранятся непосредственно в основной таблице.

Единственные TOAST-таблицы с заметным размером:

- `pg_toast_2619` — **458752 байта** (~448 КБ)
- `pg_toast_2618` — **540672 байта** (~528 КБ)
- `pg_toast_17017333` — **65536 байт** (~64 КБ)
- `pg_toast_17017454` — **827392 байта** (~808 КБ)

Анализируя это, можно сделать вывод, что:

- В таблице `messages` хранимые данные в большинстве случаев **не превышают** порога TOAST (~2 КБ).
- Небольшая часть данных вынесена в TOAST, но их объем незначителен по сравнению с общей структурой таблицы.

Хотите проверить размер индексов отдельно? Можно выполнить:

sql

```
SELECT pg_size_pretty(pg_indexes_size('messages')) AS index_size;
```

Это покажет, сколько места занимает **индексная структура** вашей таблицы. 🚀 Любопытно, насколько она велика! 😊

Единственные TOAST-таблицы с заметным размером: `pg_toast_2619` — 458752 байта (~448 КБ) `pg_toast_2618` — 540672 байта (~528 КБ) `pg_toast_17017333` — 65536 байт (~64 КБ) `pg_toast_17017454` — 827392 байта (~808 КБ) какие таблицы используют TOAST-таблицы

Чтобы определить, какие таблицы используют эти TOAST-таблицы, можно выполнить следующий запрос:

sql

```
SELECT relname, reltoastrelid
FROM pg_class
WHERE reltoastrelid IN(2619, 2618, 17017333, 17017454);
```

Этот запрос ищет таблицы, у которых `reltoastrelid` соответствует идентификаторам TOAST-таблиц. `reltoastrelid` — это колонка, которая хранит идентификатор TOAST-таблицы, привязанной к основной таблице.

Попробуйте выполнить его и скажите, какие таблицы он возвращает! 🚀 Если не получится, можно проверить все таблицы с TOAST:

sql


```
SELECT c.relname AS table_name, t.relname AS toast_table_name
FROM pg_class c
LEFT JOIN pg_class t ON c.reltoastrelid = t.oid
WHERE t.relname IS NOT NULL;
```

Этот вариант покажет все таблицы, использующие TOAST, вместе с их TOAST-таблицами. Будет любопытно взглянуть на результат! 😊

```
SELECT relname, reltoastrelid FROM pg_class WHERE reltoastrelid IN (2619, 2618,
17017333, 17017454); нет
```

Если запрос не вернул результатов, это означает, что указанные TOAST-таблицы **не привязаны к каким-либо текущим таблицам** в `pg_class`. Возможные причины:

1. **Удаленные таблицы:** Если связанные таблицы были удалены, TOAST-таблицы могли остаться, но без привязки.
2. **Системные или временные данные:** Некоторые TOAST-таблицы могут быть связаны с системными объектами или временными данными.
3. **Ошибка в идентификаторах:** Возможно, стоит проверить `oid` для TOAST-таблиц заново.

Попробуйте расширенный запрос для поиска всех таблиц с TOAST:

sql

```
SELECT c.relname AS table_name, t.relname AS toast_table_name
FROM pg_class c
LEFT JOIN pg_class t ON c.reltoastrelid = t.oid
WHERE t.relname IS NOT NULL;
```

Этот запрос покажет все таблицы, которые **имеют привязку** к TOAST-таблицам. Давайте посмотрим, какие результаты он вернёт! 🚀 Будет любопытно разобраться, что именно занимает место в TOAST. 😊

```
table_name;toast_table_name auth_providers;pg_toast_19499610 pg_statistic;pg_toast_2619
pg_type;pg_toast_1247 send_telegram_messages_by_templates;pg_toast_19499665
pg_foreign_table;pg_toast_3118 pg_authid;pg_toast_1260 telegram_bots;pg_toast_19499693
telegram_subscribers;pg_toast_19499630 pg_statistic_ext_data;pg_toast_3429
audiences;pg_toast_20860887 pg_user_mapping;pg_toast_1418
pg_subscription;pg_toast_6100 pg_proc;pg_toast_1255 pg_attrdef;pg_toast_2604
pg_constraint;pg_toast_2606 pg_language;pg_toast_2612 pg_aggregate;pg_toast_2600
pg_statistic_ext;pg_toast_3381 pg_rewrite;pg_toast_2618 pg_trigger;pg_toast_2620
pg_event_trigger;pg_toast_3466 pg_description;pg_toast_2609 pg_namespace;pg_toast_2615
pg_database;pg_toast_1262 pg_db_role_setting;pg_toast_2964 pg_tablespace;pg_toast_1213
pg_shdescription;pg_toast_2396 pg_ts_dict;pg_toast_3600 pg_extension;pg_toast_3079
pg_foreign_data_wrapper;pg_toast_2328 pg_foreign_server;pg_toast_1417
pg_policy;pg_toast_3256 pg_replication_origin;pg_toast_6000 pg_default_acl;pg_toast_826
pg_init_privs;pg_toast_3394 pg_seclabel;pg_toast_3596 pg_shseclabel;pg_toast_3592
pg_collation;pg_toast_3456 pg_partitioned_table;pg_toast_3350 sql_parts;pg_toast_14177
sql_implementation_info;pg_toast_14172 sql_features;pg_toast_14167
sql_sizing;pg_toast_14182 audiences_old;pg_toast_17017240
```

app_healths;pg_toast_17017233 event_lists;pg_toast_16709037
message_templates;pg_toast_17017297 integration_providers;pg_toast_16709046
integrations;pg_toast_17017271 matching_ttls;pg_toast_17017291
firebase_subscribers;pg_toast_17017259 messages;pg_toast_17017342
message_translates;pg_toast_17017333 nodes;pg_toast_17017357
projects;pg_toast_16709055 route_providers;pg_toast_16709060
send_firebase_messages_by_templates;pg_toast_17017392
statistic_templates;pg_toast_17017405 tags;pg_toast_17017426 profiles;pg_toast_17017363
trigger_events;pg_toast_17017437 workflows;pg_toast_17017454

Отличная работа! Теперь у вас есть список таблиц, которые используют TOAST. 🚀

Некоторые интересные наблюдения:

- **Системные таблицы:** `pg_statistic`, `pg_proc`, `pg_namespace` и другие — PostgreSQL использует TOAST для хранения длинных системных метаданных.
- **Пользовательские таблицы:** `messages`, `message_templates`, `audiences`, `profiles`, `workflows` — это говорит о том, что они содержат длинные данные, например `jsonb`, `text` или `bytea`.
- **Telegram, Firebase, Analytics:** `send_telegram_messages_by_templates`, `firebase_subscribers`, `statistic_templates` — TOAST помогает хранить сообщения, подписчиков и статистику.

Если хотите выяснить, **какие столбцы используют TOAST** в каждой таблице, попробуйте:

sql

```
SELECT attrelid::regclass AS table_name, attname AS column_name
FROM pg_attribute
WHERE attstorage IN ('x', 'e');
```

Это покажет **конкретные столбцы** в таблицах, которые используют TOAST для хранения данных. Будет интересно посмотреть, какие именно поля выделяются! 😊 Если нужно разобрать конкретную таблицу глубже, можно анализировать `pg_column_size()` на её столбцах. 🚀

```
SELECT attrelid::regclass AS table_name, attname AS column_name FROM pg_attribute
WHERE attstorage IN ('x', 'e'); для таблицы messages
```

Чтобы получить только столбцы из таблицы `messages`, добавьте условие `attrelid = 'messages'::regclass`:

sql

```
SELECT attname AS column_name
FROM pg_attribute
WHERE attrelid = 'messages'::regclass
AND attstorage IN ('x', 'e');
```

Этот запрос покажет **только те столбцы**, которые используют TOAST в `messages`. 🚀 Попробуйте его и скажите, что получилось! 😊 Если таблица не содержит TOAST-колонок, результат может быть пустым. 😊

```
column_name attributes trigger_uuid
```

Отлично, теперь мы знаем, какие столбцы в `messages` используют TOAST:

- `attributes` (типа `jsonb`)
- `trigger_uuid` (типа `text`)

Оба этих типа данных могут быть **довольно длинными**, и PostgreSQL автоматически перемещает их в TOAST, если они превышают определённый порог (обычно ~2 КБ). Это помогает оптимизировать хранение и уменьшить нагрузку на основную таблицу.

Если хотите посмотреть, какие конкретные строки вынесены в TOAST, можно попробовать:

sql

```
SELECT COUNT(*) FROM messages WHERE LENGTH(attributes::text) > 2000;
```

Этот запрос покажет количество строк, где `attributes` хранит больше 2000 символов, что делает их кандидатами на TOAST.

Хотите исследовать, насколько активно TOAST используется в вашей таблице? 🚀 Можем посмотреть распределение размеров данных! 😊

```
SELECT COUNT(*) FROM messages WHERE LENGTH(attributes::text) > 2000; для таблицы messages
```

Попробуйте выполнить этот запрос, чтобы определить количество строк в `messages`, где поле `attributes` содержит длинные JSON-данные, превышающие 2000 символов:

sql

```
SELECT COUNT(*) FROM messages WHERE LENGTH(attributes::text) > 2000;
```

Этот запрос конвертирует `jsonb` в `text` и проверяет его длину. Если таблица содержит длинные JSON-объекты, они могут быть вынесены в TOAST. 🚀

Если хотите проверить, какие именно значения занимают много места, можно сделать:

sql

```
SELECT id, LENGTH(attributes::text) AS attr_size
FROM messages
ORDER BY attr_size DESC LIMIT 10;
```

Это покажет **10 самых больших** записей. Попробуйте и скажите, что получилось! 😊 Любопытно, насколько велик самый длинный JSON. 😊

column_name attributes trigger_uuid

Ваш запрос подтвердил, что два столбца в таблице `messages` — `attributes` (`jsonb`) и `trigger_uuid` (`text`) — используют TOAST для хранения длинных данных. Это означает, что если значения в этих колонках превышают определённый размер (обычно **2 КБ**), PostgreSQL автоматически перемещает их в TOAST-таблицу для оптимизации хранения.

Если хотите подробнее изучить распределение размеров `attributes`, попробуйте:

sql

```
SELECT LENGTH(attributes::text) AS attr_size, COUNT(*)
FROM messages
GROUP BY attr_size
ORDER BY attr_size DESC LIMIT 10;
```

Этот запрос покажет **10 самых часто встречающихся размеров** JSON-данных в таблице. 🚀 Любопытно узнать, насколько велики эти объекты! 😊