



AXE

(Аналитический акселератор)

Марк Ривкин

Начальник отдела технического консалтинга,
Постгрес Про

m.rivkin@postgrespro.ru



Типы систем

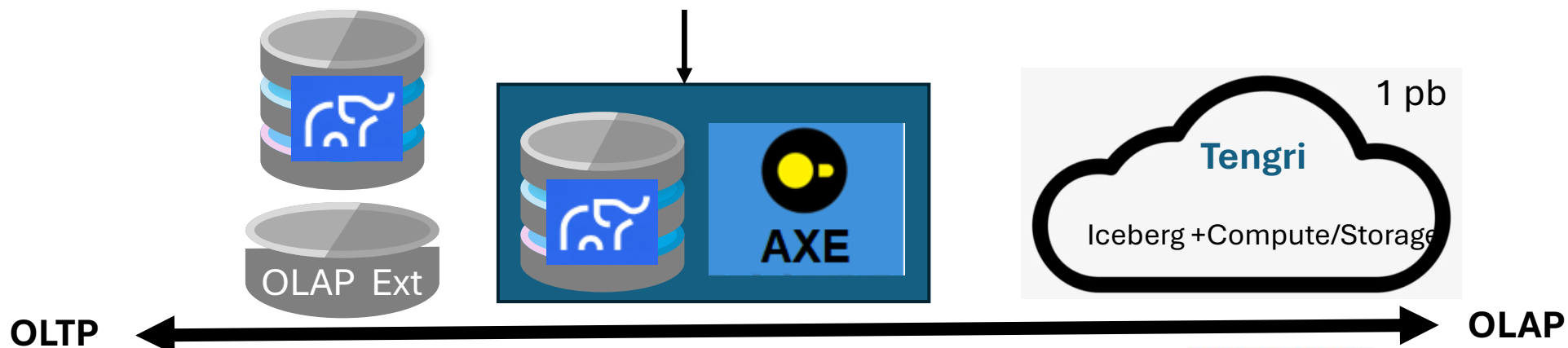
- **OLTP - Системы для обработки большого количества коротких транзакций в реальном времени.**
 - Оптимизированы для CRUD-операций (Create, Read, Update, Delete)
 - Высокая доступность и скорость отклика
 - Нормализованные структуры данных
 - ACID-требования (Atomicity, Consistency, Isolation, Durability)
 - Горизонтальное масштабирование через шардинг
 - Репликация для отказоустойчивости
- **Аналитические системы (OLAP) - Системы для сложного анализа больших объемов исторических данных.**
 - Оптимизированы для чтения и агрегации
 - Денормализованные структуры (звезда, снежинка)
 - Пакетная обработка данных
 - Редкие изменения

OLTP, OLAP, HTAP

Критерий	OLTP (транзакционные)	OLAP (аналитические)
Цель	Оперативная обработка	Анализ исторических данных
Тип запросов	Короткие, простые	Сложные, агрегирующие
Данные	Актуальные	Исторические (годы)
Пример	Банковская транзакция	Отчет по продажам за 5 лет

В реальной жизни часто видим смесь нагрузок - HTAP

Можно иметь 2 разные СУБД (для OLTP и аналитики, но это дорого и аналитика отстает от OLTP)
Можно сделать универсальную/конвергентную СУБД (как Oracle) и работать на свежих данных



CFS, ILM
 Partitioning
 Parallel
 Analytical func
 In-memory
 columnar
 (HIMERA)
 Global Index
 Result cache
 Optimizer
 iHeap table
 Append optimized
 table

Postgres



PP Analytical Accelerator (Pgpro_AXE):

Pgpro_axe
 Pgpro_metastore



**OLTP и OLAP,
 DWH, HTAP**

OLTP и большие OLAP

Очень большие OLAP

Типы аналитиков

- Традиционные - АХЕ
- Новые (SQL + Python) - Тенгри
- Работающие с горячими данными - НИМЕРА
- Есть еще разработчики, архитекторы, СІО и т д - АХЕ

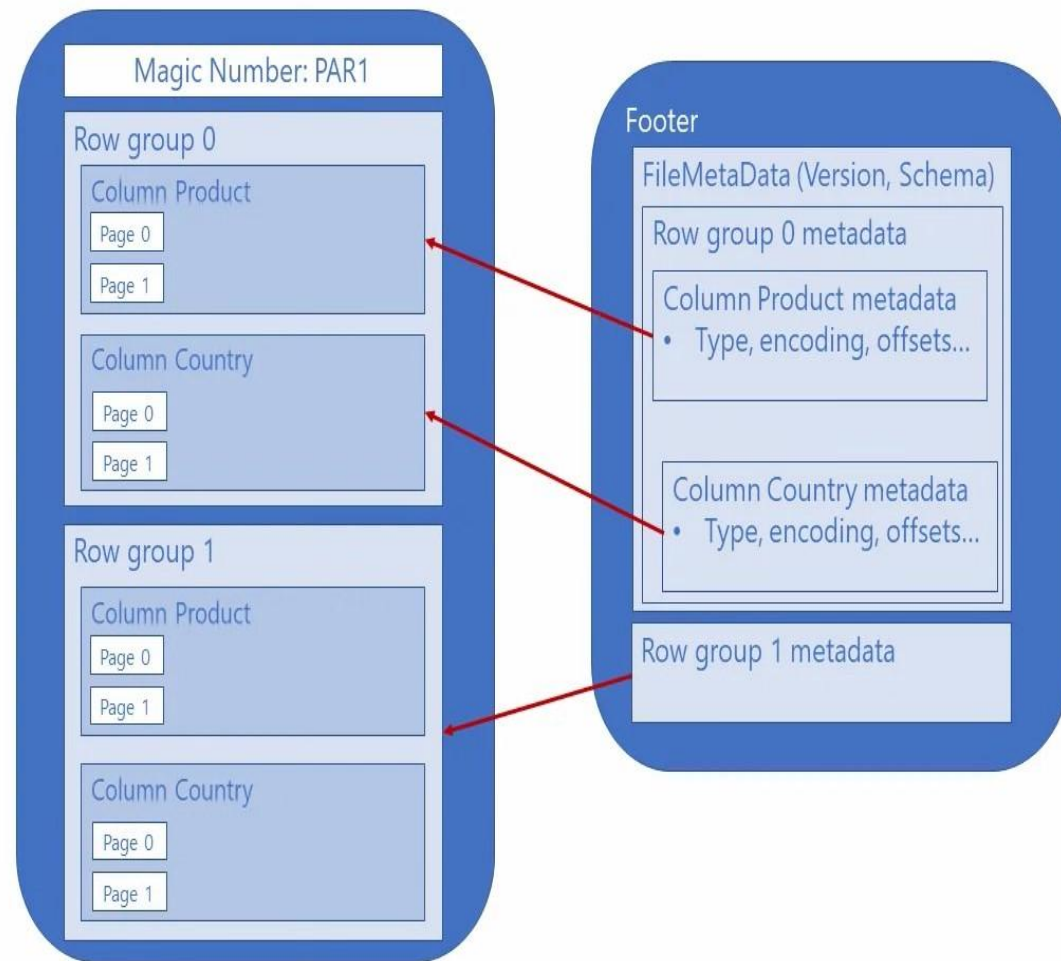
Postgres Pro Analytical Accelerator (AXE):

Что такое АХЕ ?

- Это акселератор (ускоритель)
- Волшебная кнопка на авто/мотоцикле, ускоряющая в 10-50 раз на спец дороге (аналитика)
- Очень проста в установке и использовании с Postgres Pro (2 расширения)
- Знакома всем пользователям Postgres – продавать им легко

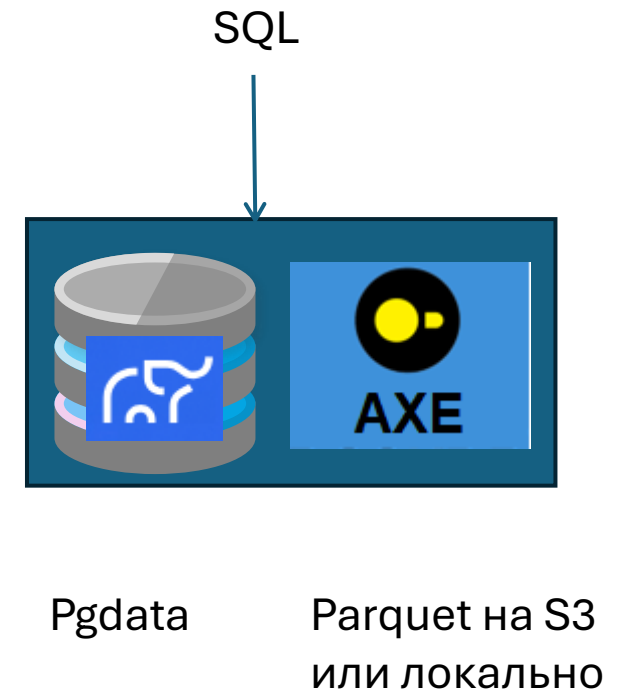
Postgres Pro Enterprise + AXE

- Структура Parquet идеальна для OLAP
- Паркетные файлы быстры т к читаются не все строки и не все колонки + сжатие – т е малый I/O (сжатие в 50 раз в пилоте)
- SIMD и параллелизм
- Файлы parquet создаются из таблиц Postgres с помощью утилиты ProCopy или команды COPY
- 10 млн записей преобразуются в Parquet за 8 сек
- Можно автоматически строить view в Postgres Pro, ссылающуюся на Parquet

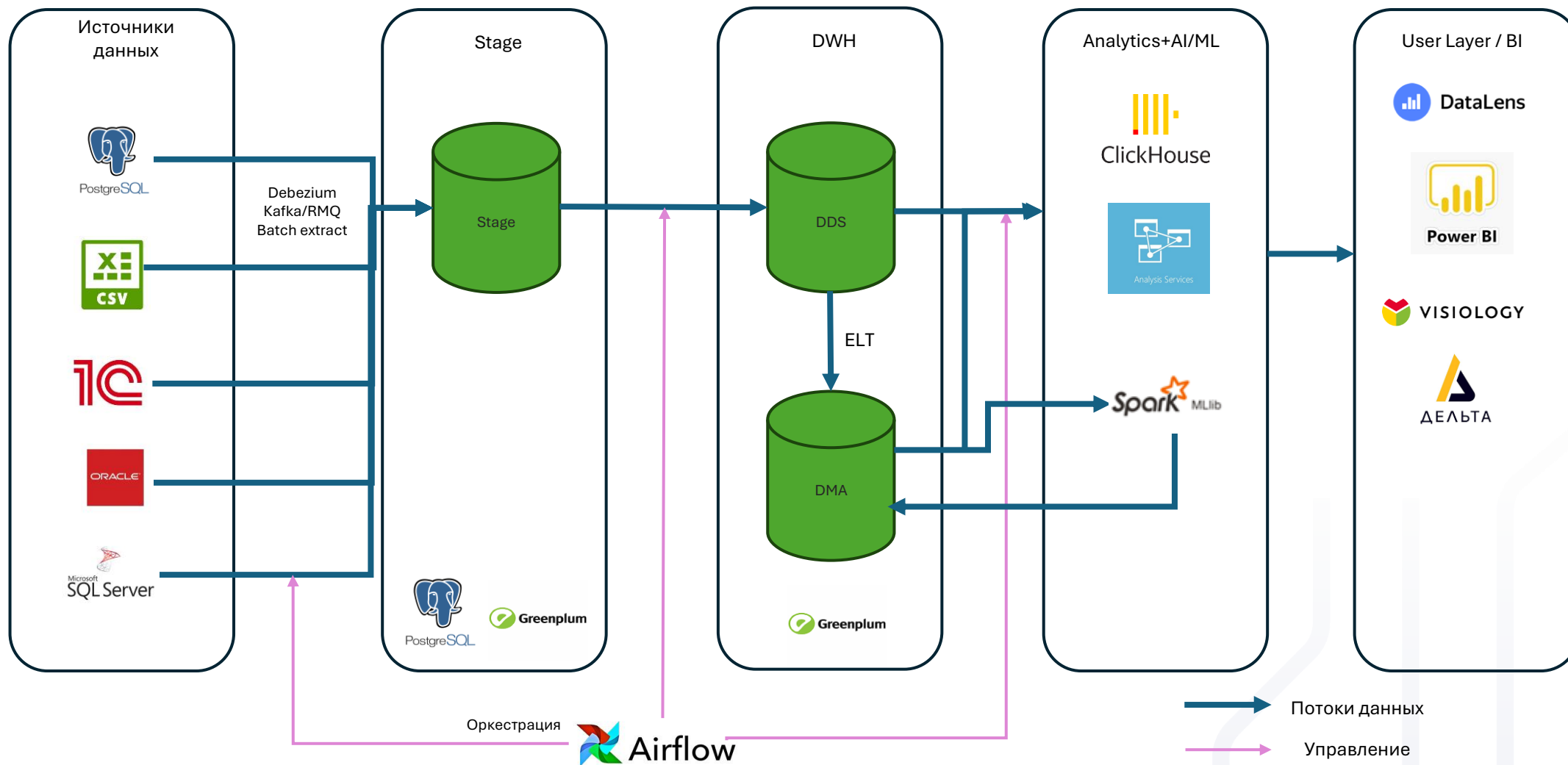


Postgres Pro Enterprise + pgpro_AXE

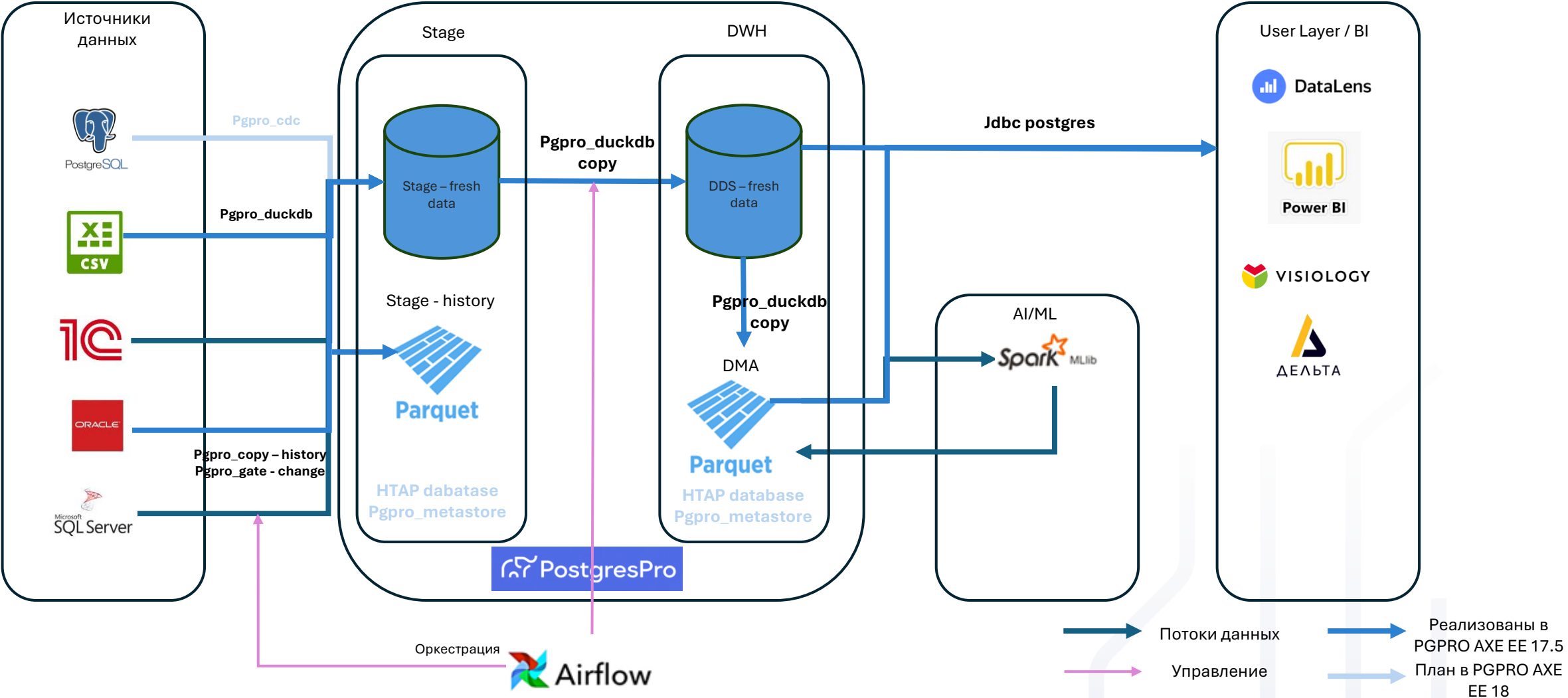
- Pgpro_axe - расширение к Postgres Pro Enterprise
- На том же компьютере, что и Postgres Pro
- Единая точка входа и единый SQL
- Таблицы Postgres Pro хранятся как обычно, данные AXE – в паркетных файлах на S3 или локально
- Запрос может быть к тем или иным таблицам или смешанный
- 2 оптимизатора запросов
- Для масштабирования аналитики можно создавать доп узлы с пустой БД, они работают только с parquet на S3 (100Tb * x с S3)



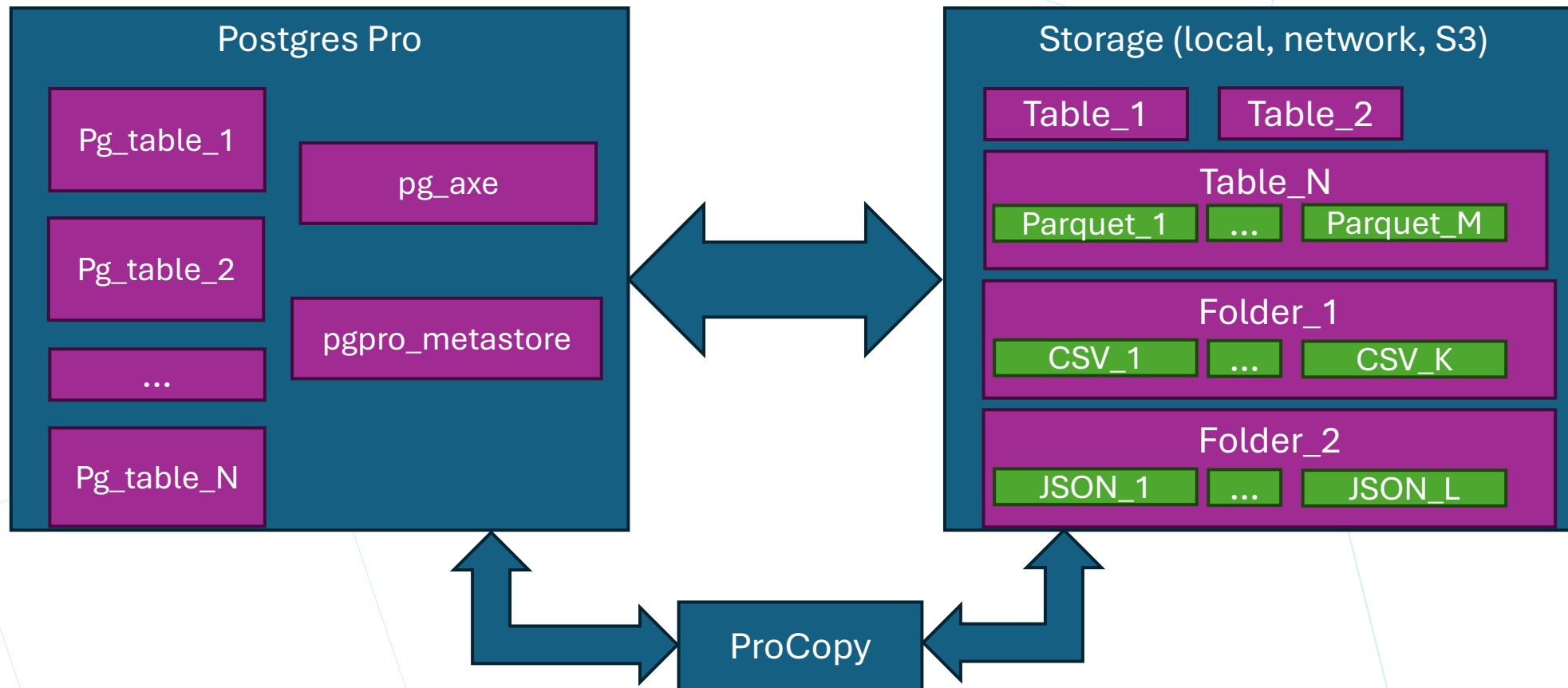
Типовой OLAP ландшафт



OLAP ландшафт на основе PGPRO AXE



Архитектура



Набор компонент платформы

Расширения Postgres

pgpro_axe

Векторный движок Postgres Pro с поддержкой разных систем хранения (NVME, NFS, S3) и Parquet

(01)

pgpro_metastore

Каталог файлов аналитических таблиц и средства репликации таблиц Postgres

(02)

Pgpro Copy — репликация из дополнительных источников

Расширение pgpro_ахе

Векторный движок Postgres Pro:

1. Быстрые запросы над колоночными источниками данных (Parquet файлы и временные колоночные таблицы)

3. Объединение данных HEAP таблиц Postgres и колоночных источников

2. Практически полная поддержка синтаксиса Postgres

4. Поддержка работы с объектными хранилищами (S3)

Pgpro_metastore

- В версии 1 паркет только на чтение, потом CDC
- Pgpro_metastore - расширение к Postgres Pro Enterprise, аналог Apache Iceberg, но хранит каталог с метаданной в БД
- Metadata Layer хранит информацию о всех файлах паркетной таблицы
- Можно добавлять партиции в parquet таблицу
- Интеграция с Pgpro_backup
- Ролевая модель
- ACID

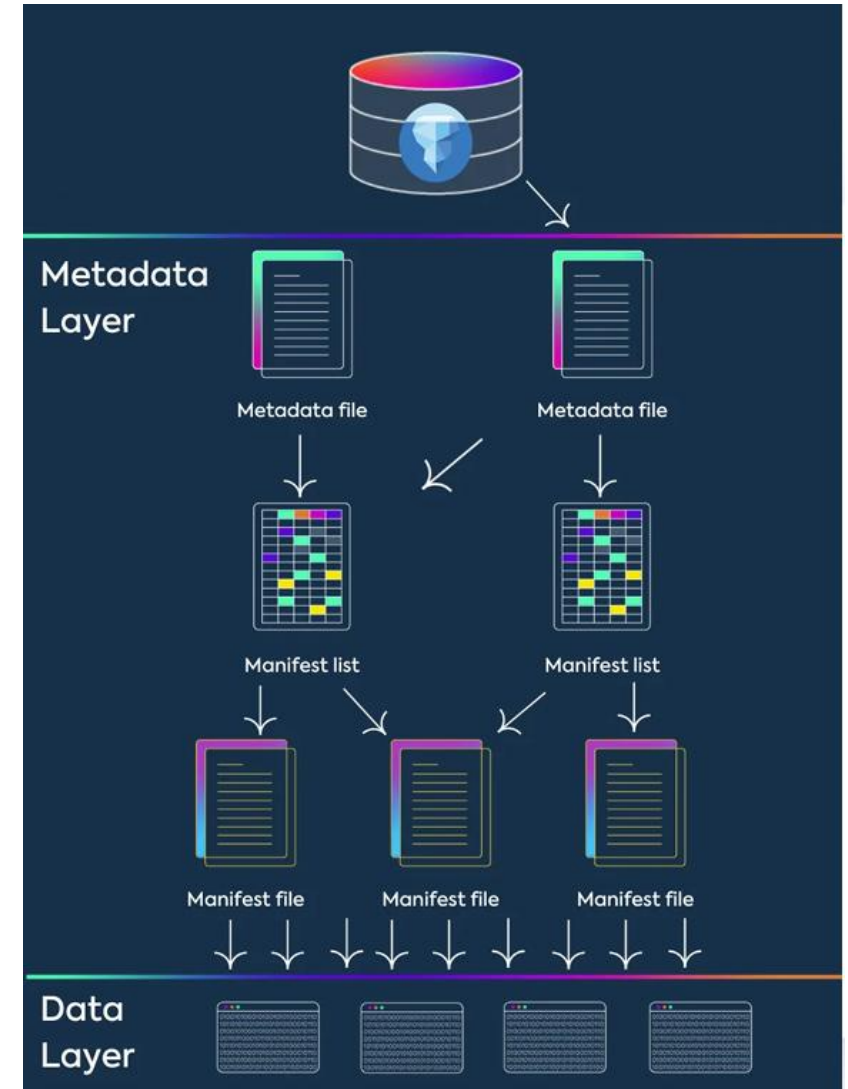
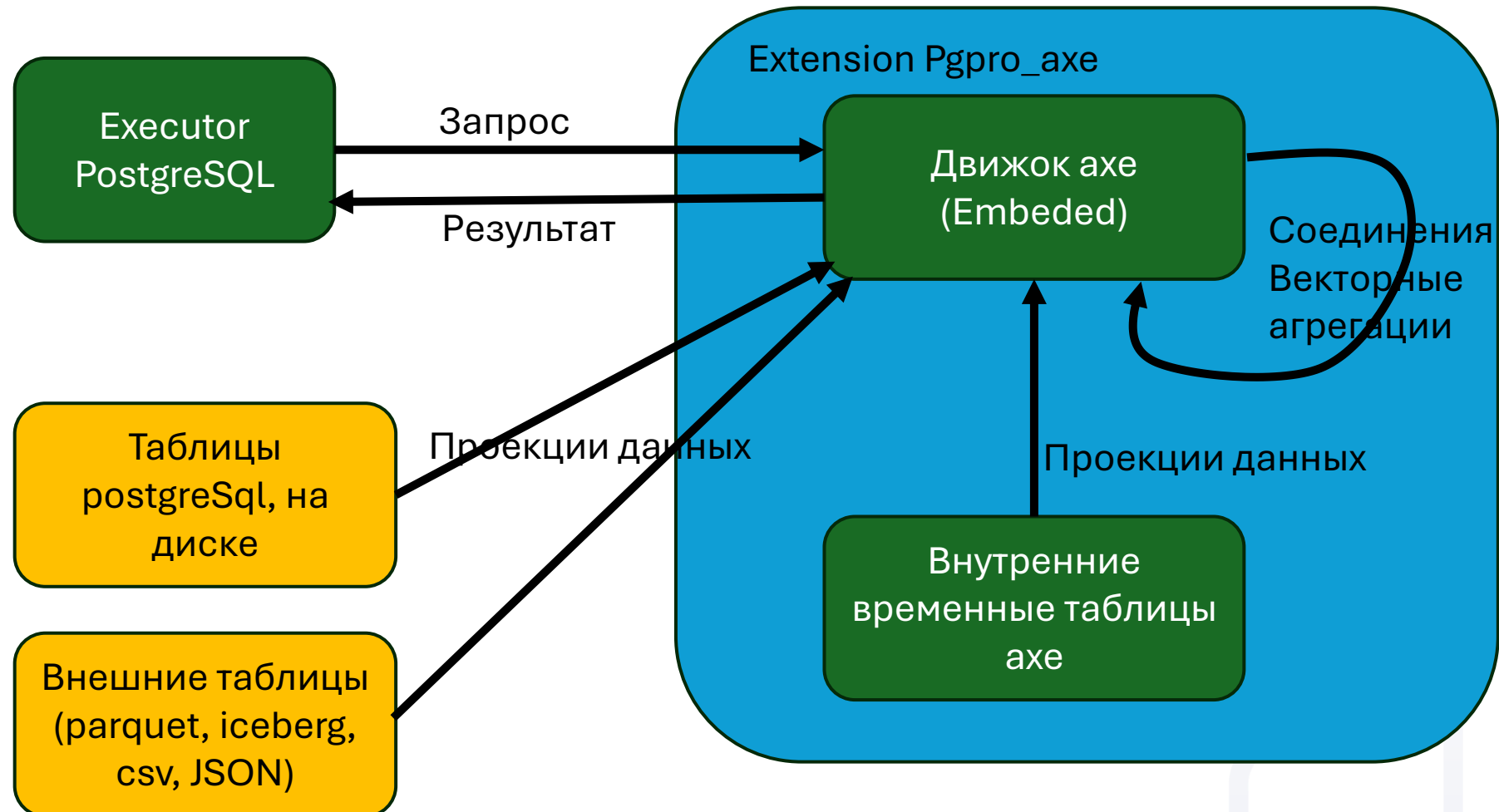


Схема работы pgpro_ахе

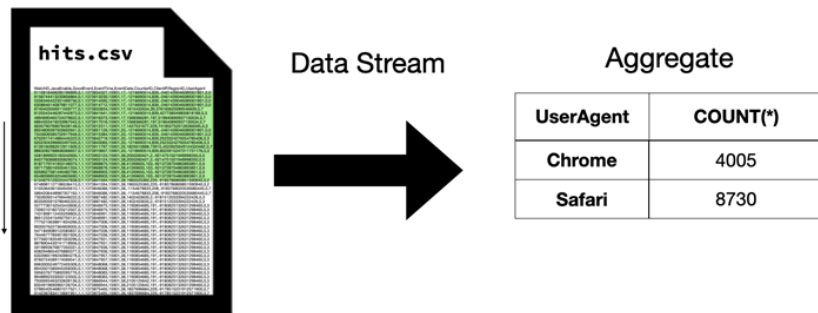


Ключевые особенности архитектуры AXE

1. Pgpro_axe наше расширение на основе pg_duckdb
2. Встраиваемая колоночная СУБД, оптимизированная для аналитики (OLAP).
3. Написана на C++ (для скорости).
4. Есть свои внутренние таблицы - в RAM, и внешние – формат parquet, Iceberg, DeltaLake и др.
5. Векторные вычисления блоками по 2048 значения, а не по одной строке. Используется L1 кэш CPU и SIMD инструкции.
6. Организация данных в RAM – векторный формат Arrow.
7. Поддерживает многопоточность
8. Интерфейсы – SQL движок, API для Python, R, Java, C++, Rust и др.

Особенности выполнения запросов

Streaming execution: Данные подгружаются в RAM порциями, без необходимости выгрузки всего датасета для расчетов.

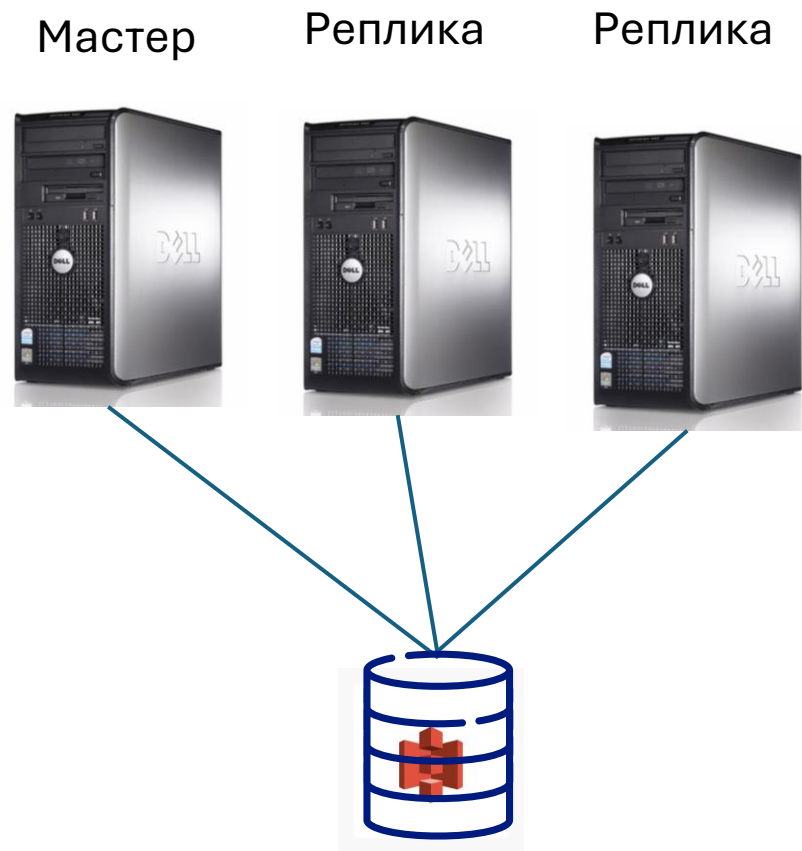


Чтение файлов и запись на диск также идут в параллель и работают порционно. Промежуточные данные могут выгружаться на диск – spilling. Данные из присоединенных БД также подгружаются в память до заполнения лимита. Разные запросы к тем же данным показывают лучше производительность.

Мощный собственный оптимизатор (Filter Pushdown, Join Order Optimization, TopN Optimization), нет проблем в сложных аналитических запросах (в т.ч. для TPC-H, TPC-DS)

Масштабирование

- На мастере
- На реплике
- На группе реплик с доступом к единому S3 (100 Tb * X)

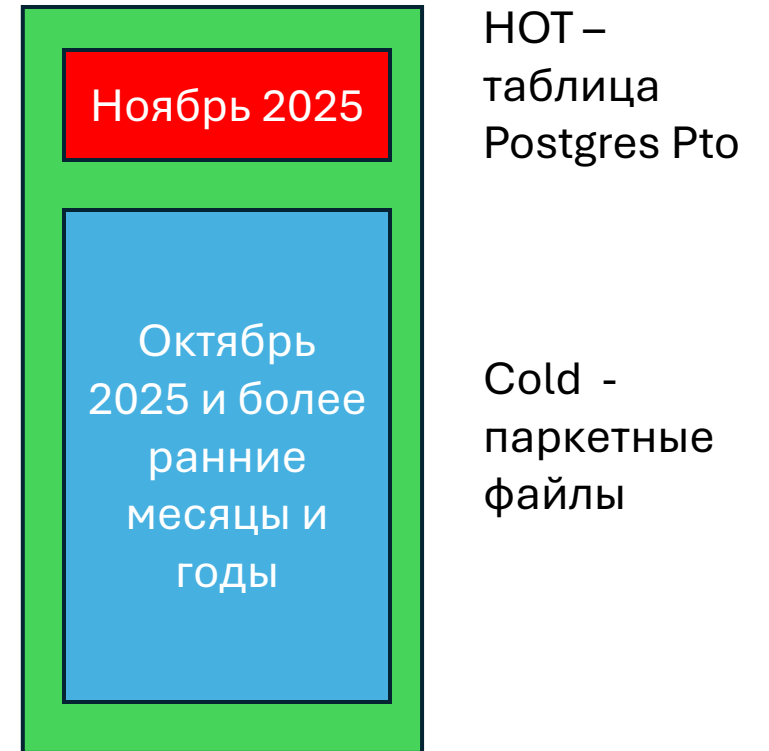


S3 Object storage с Parquet

Работа с теплыми и горячими данными

- Перелив по ночам
- Добавление порции данных
- Горячие данные в Postgres (текущий месяц), холодные – в АХЕ и единый VIEW на всю таблицу

View = hot union cold



Результаты тестов

- Для ClickHouse, AXE, Postgres Pro - компьютер с 8 vCPU, 16 Гб RAM
- Для CITUS и NN – гораздо больше
- 50 миллионов записей (2 Tb база)
- `Select * from read_parquet('/data/...../file_pf_50_1*') r where r['name'] = ;`
- Время в секундах

Параметры	AXE	ClickHouse	Citus	Postgres Pro
Время операции 1	0,7	0,37	8,77	30,01
Время операции 2	0,68	0,64	9,29	42,31

```
Select r['file_name'], r['row_group_size'], r['statistics']  
From duckdb_query('FROM parquet_metadata("/data/.....file1.parquet")') r;
```

Postgres Pro без и с AXE

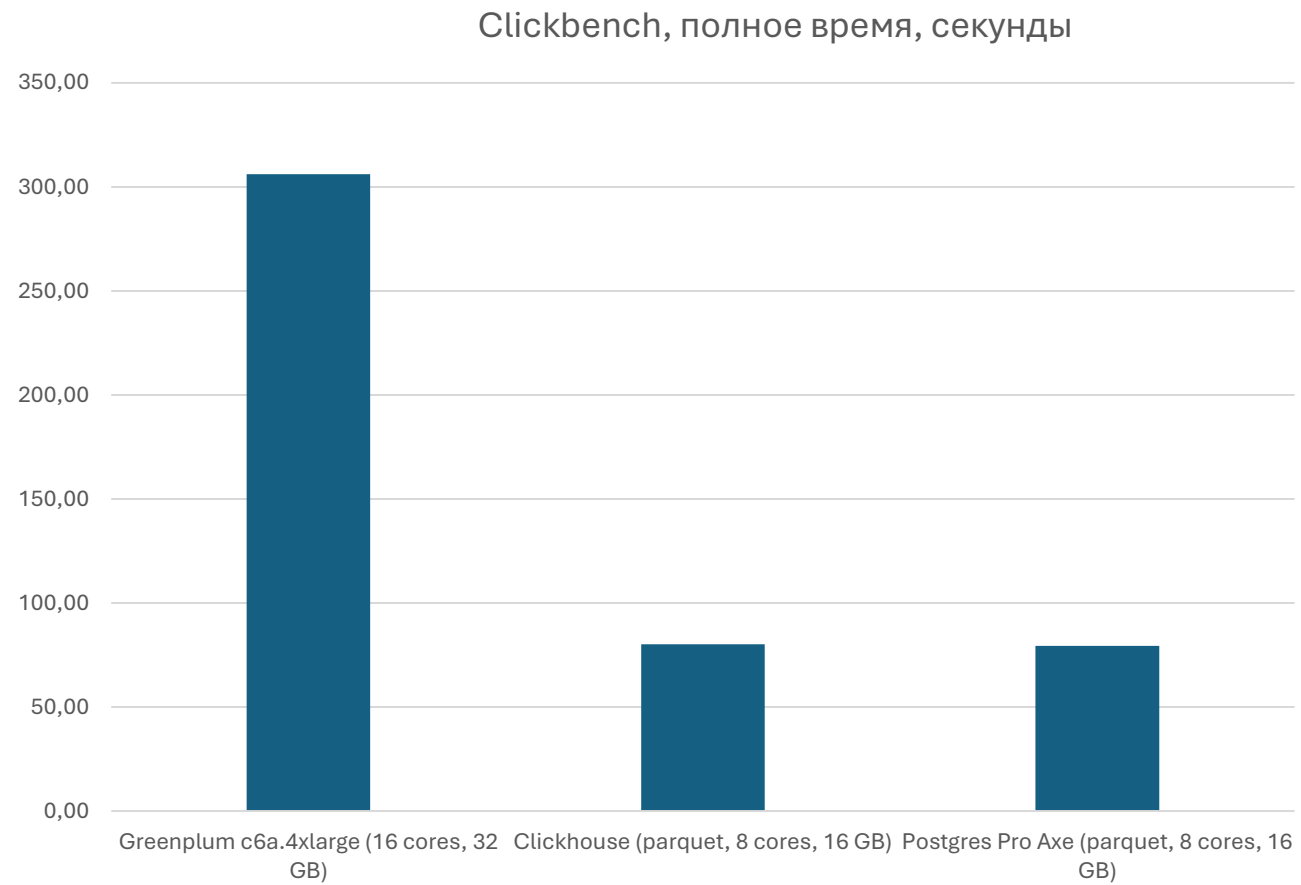
64 ядра, 64 Гб RAM, 6 млн строк

Запрос N	PG tables	AXE	Ускорение в
1	7.4186s	0.128s	57
2	4.0139s	0.175s	22
3	1.6309s	0.0373s	43
4	1.9727s	0.0423s	46
5	1.5942s	0.0359s	44
6	1.9371s	0.0346s	55
7	3.4457s	0.0713s	48
8	5.2353s	0.167s	31
9	1.6342s	0.0567s	28
10	2.0166s	0.0426s	47
11	1.6746s	0.0408s	41

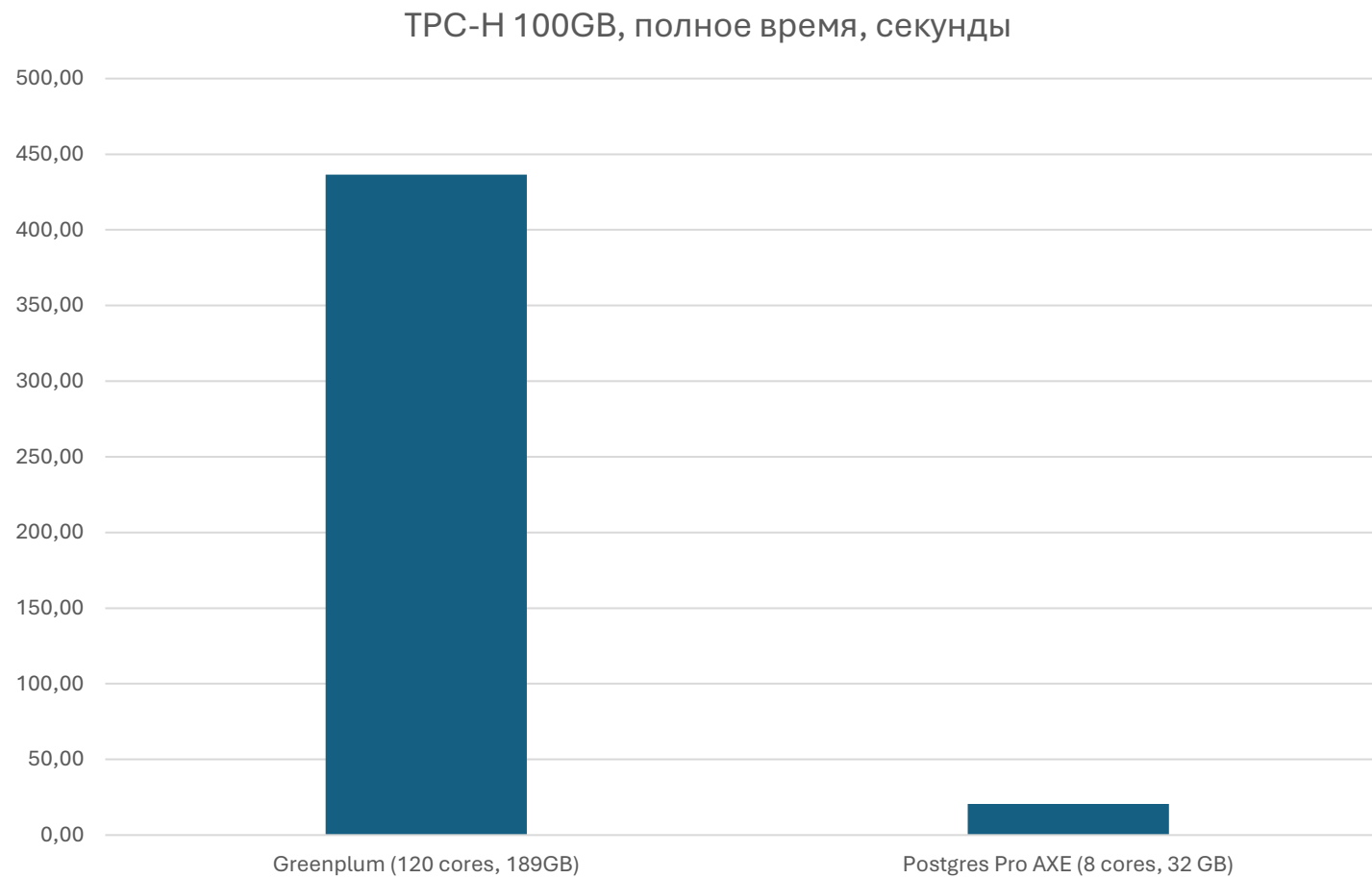
Тесты другого заказчика

#	Результаты	PG heap 6 cores * 36 GB	AXE with SIMD 6 cores * 36 GB
1	Запрос актуальных данных для Хаба 1 (Вариант 1: DISTINCT ON)	3.585s	0.344s
2	Запрос актуальных данных для Хаба 1 (Вариант 2: LATERAL JOIN)	3.486s	0.984s
3	Запрос актуальных данных для Хаба 1 (Вариант 3: Подзапросы с MAX(load_date))	11.342s	0.818s
4	Запрос актуальных данных для Хаба 1 (Вариант 4: Подзапросы с MAX(load_date))	11.933s	0.804s
5	Запрос актуальных данных для Хаба 1 (Вариант 5: ROW_NUMBER)	4.230s	0.692s
6	Запрос актуальных данных для Хаба 1 (Вариант 6: CTE с агрегацией)	16.283s	0.274s
7	Запрос актуальных данных для Хаба 1 (Вариант 7: EXISTS)	6.426s	0.798s
8	Запрос актуальных данных для Хаба 1 (Вариант 8: Подзапросы с LIMIT 1)	3.310s	0.999s
9	Запрос актуальных данных для Хаба 1 (Вариант 9: LATERAL JOIN с ROW_NUMBER)	5.045s	0.962s
10	Запрос актуальных данных для Хаба 1 (Вариант 10: FIRST_VALUE)	4.997s	1.42s
11	Запрос полной истории для Хаба 1 (Вариант 1: LATERAL JOIN с подзапросами)	32.111s	5.93s
12	Запрос полной истории для Хаба 1 (Вариант 2: FIRST_VALUE)	25.192s	3.32s
13	Запрос полной истории для Хаба 1 (Вариант 3: ORDER BY/LIMIT)	27.029s	5.00s
14	Запрос полной истории для Хаба 1 (Вариант 4: EXISTS с подзапросами)	49.087s	9.51s
15	Запрос полной истории для Хаба 1 (Вариант 5: UNION ALL + FIRST VALUE)	1m:20.278s	14.06s
16	Запрос на точку во времени для Хаба 1 (Вариант 1: Подзапросы)	3.918s	0.637s
17	Запрос на точку во времени для Хаба 1 (Вариант 2: DISTINCT ON)	1.887s	0.262s
18	Запрос на точку во времени для Хаба 1 (Вариант 3: ROW_NUMBER)	2.866s	0.380s
19	Запрос на точку во времени для Хаба 1 (Вариант 4: EXISTS)	2.921s	0.390s

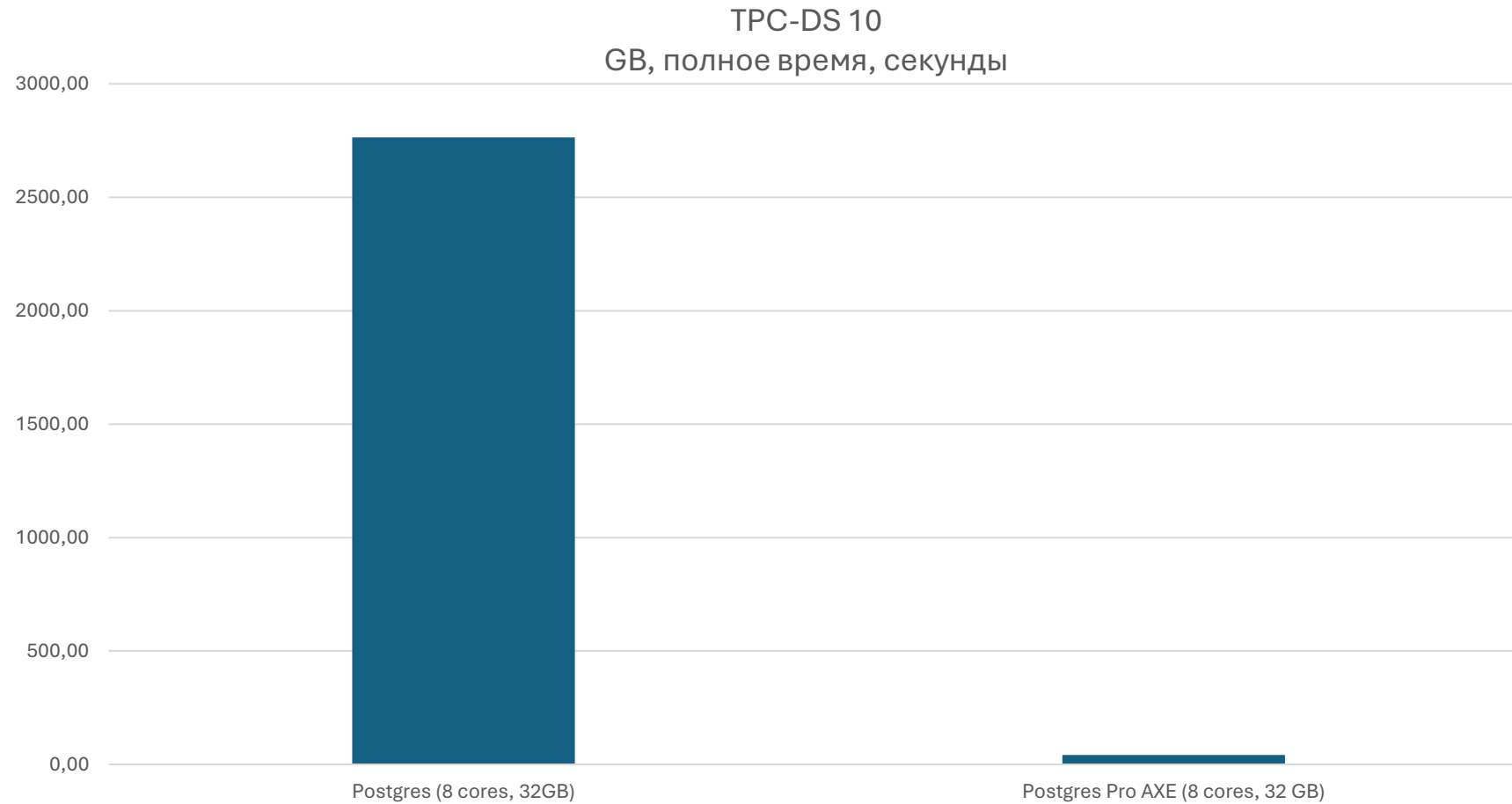
Тесты Clickbench



Тесты TPC-H



Тесты TPC-DS



Примеры команд (AXE):

Задание параметров акселератора

select name, setting **from** pg_settings **where** name **like** '%duckdb%'; - параметры
pgpro_duckdb

alter system set duckdb.max_temp_directory_size='40GB' – размер временных файлов
duckdb.max_memory , duckdb.memory_limit – размер RAM.

После изменения – рестарт сервера PG.

SELECT * FROM duckdb.query('SELECT name, value FROM duckdb_settings()'); - параметры
встроенной базы duckdb

Отличия query и raw_query

Query – выполняет команду в pg_ахе, использует парсер ахе выводит результат в результаты Postgres. Используется для выполнения команды Select. В основном для выполнения функций АХЕ (например, duckdb_settings, parquet_schema, parquet_metadata и т д)

Raw_query – отправляет команду напрямую в ахе, пишет результат в output. Используется для DDL и DML операций внутри Акселератора.

Команды Raw query

Select, left/right/full join

explain analyze, explain

DDL

DML для временных таблиц

Postgres Pro АХЕ: запросы к группе файлов (*)

Выборка из Parquet файлов

```
SELECT
  r['item_id'],
  r['product_name'],
  r['status']
FROM read_parquet ('/dir/items/*') r
WHERE r['status'] = 'Новый';
```

1



```
dir
├── items
│   ├── data202511.parquet
│   ├── data202512.parquet
│   ├── data202601.parquet
│   └── data202602.parquet
```

```
SELECT
  r['item_id'],
  r['product_name'],
  r['status']
FROM read_parquet ('/dir/items/data2026*.parquet') r
WHERE r['status'] = 'Новый';
```

2



```
dir
├── items
│   ├── data202601.parquet
│   └── data202602.parquet
```


Запросы на Postgres к PG и AXE

```
explain analyze SELECT t.id, t.name, t.price,  
avg(a.doc_id), count(1)  
FROM public.t_sng_pf_50_1 a LEFT JOIN  
public.pg_my_table t  
ON a.registration_month = t.id  
group by t.id, t.name, t.price;
```

```
EXPLAIN ANALYZE SELECT t.id, t.name, t.price, avg(a.doc_id) AS avg, count(1) AS count  
id, a_1.doc_name AS doc_name, a_1.registration_date AS registration_date, a_1.registrat  
FROM system.main.read_parquet('/data/sng/t_sng_pf_50_1_*'::text) a_1) a LEFT JOIN pgduct  
gistration_month = t.id))) GROUP BY t.id, t.name, t.price
```

Total Time: 0.305s

QUERY

EXPLAIN_ANALYZE

0 Rows
(0.00s)

1. Работает через стандартный драйвер Postgres.
2. Данные передаются в аналитический движок, результат возвращается в клиентскую сессию.
3. Позволяет соединять данные Postgres и внешних файлов.
4. Работает быстрая векторная обработка и проекции колонок parquet.
5. Скорость работы – 0.3 sec / 4 CPU 4 GB RAM / 50M строк

Запись во внешние parquet файлы

Запись в parquet с заданным размером группировки строк

COPY

(select * FROM generate_series(1,100000))

TO '/data/sng/res/series1.parquet'

(FORMAT parquet, COMPRESSION zstd, ROW_GROUP_SIZE 10000, PARTITION BY year);

select * from duckdb.query('FROM parquet_metadata("/data/sng/res/series1.parquet")');

- метаданные паркет файла – 10 групп строк.

```
select r['file_name'], r['row_group_id'], r['row_group_bytes'], r['stats_min'], r['stats_max']  
from duckdb.query('FROM parquet_metadata("/data/sng/res/series1.parquet")') r;
```

ults 1 ×

select r['file_name'], r['row_group_id'], r['row_group_bytes'] Enter a SQL expression to filter results (use Ctrl+Space)

AZ file_name	123 row_group_id	123 row_group_bytes	AZ stats_min	AZ stats_max
/data/sng/res/series1.parquet	0	81,951	1	10240
/data/sng/res/series1.parquet	1	81,951	10241	20480
/data/sng/res/series1.parquet	2	81,951	20481	30720
/data/sng/res/series1.parquet	3	81,951	30721	40960
/data/sng/res/series1.parquet	4	81,951	40961	51200
/data/sng/res/series1.parquet	5	81,951	51201	61440
/data/sng/res/series1.parquet	6	81,951	61441	71680
/data/sng/res/series1.parquet	7	81,951	71681	81920
/data/sng/res/series1.parquet	8	81,951	81921	92160
/data/sng/res/series1.parquet	9	62,748	92161	100000

Postgres Pro АХЕ: функция создания паркета и view

```
SELECT
  create_view_as_parquet (
    p_table_schema => 'orders',
    p_table_name => 'items',
    p_parquet_path => '/dir/items'
  );
```



```
CREATE OR REPLACE VIEW orders_parquets.items
AS
SELECT
  r['item_id']::integer AS item_id,
  r['product_name']::text AS product_name,
  r['status']::text AS status
FROM
  read_parquet ('/dir/items/*') r;
```

Объединение данных из разных источников в одном запросе

```
SELECT
  f['route_no'],
  f['scheduled_departure'],
  t.book_ref,
  s['price']
FROM read_parquet ('/pgdata/parquet/flights.parquet') f -- Parquet файл на локальном диске сервера
JOIN read_parquet ('s3://demo/segments.parquet') s ON s['flight_id'] = f['flight_id'] -- Parquet файл на S3
JOIN bookings.tickets t ON t.ticket_no = s['ticket_no'] -- HEAP таблица
WHERE f['route_no'] = 'PG0100';
```



route_no	scheduled_departure	book_ref	price
PG0100	2028-02-22 16:35:00+03	L3C8AZ	6250.00
PG0100	2028-02-22 16:35:00+03	E5GSVY	6250.00

...

Пользовательские функции PG в запросах AXE

-- Создаем пользовательскую функцию в PG

```
CREATE OR REPLACE FUNCTION demo_convert_to_cny (p_amount numeric)
RETURNS numeric
LANGUAGE plpgsql
AS $function$
BEGIN
...
RETURN ...;
END;
$function$
```

-- Запускаем запрос в AXE и копируем результат во временную таблицу PG

```
CREATE TEMPORARY TABLE tmp_report AS
SELECT s['ticket_no'], s['price'] FROM read_parquet ('s3://demo/segments.parquet') s
WHERE ...;
```

-- Применяем пользовательскую функцию на временной таблице

```
SELECT sum(price) amount_rub,
       sum(demo_convert_to_cny (price, 11.2)) amount_cny
FROM tmp_report;
```

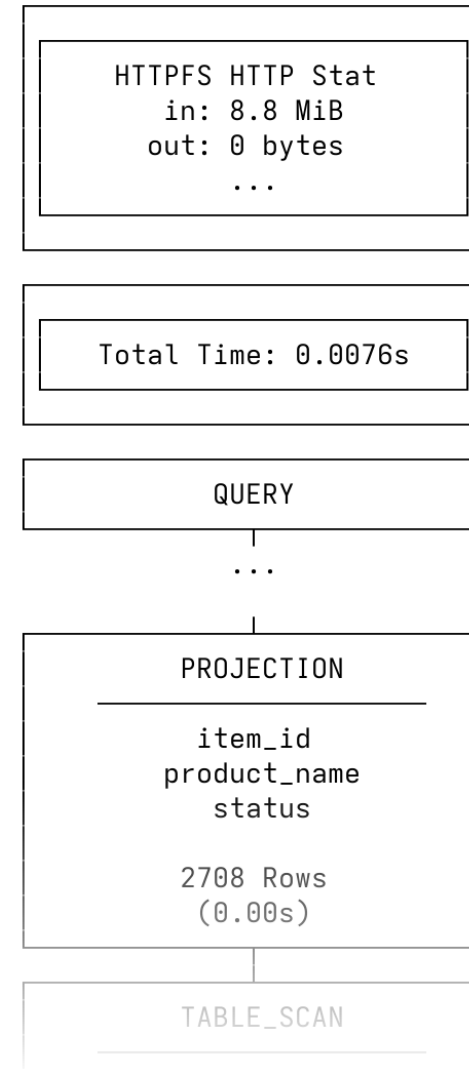


В будущем будет реализовано управление планом запроса с целью разделения:

- одних узлов плана в **PG**
- других узлов плана в **AXE**

Postgres Pro AXE: планы запросов

```
EXPLAIN ANALYZE
SELECT
  r['item_id'],
  r['product_name'],
  r['status']
FROM read_parquet ('/dir/items/*') r
WHERE r['status'] = 'Новый';
```



Сценарии использования PGPRO AXE

1. Копирование данных OLTP базы Postgres или реплики в parquet, перенастройка отчетов на получившиеся файлы.
 - Скорость OLAP расчетов ускоряется в 30 и более раз по сравнению с HEAP таблицами. Используются те же ресурсы Postgres сервера, но в кратно меньшем объеме, Сжатие данных в 2-7 раз по сравнению со строковым хранением.
2. Аналитика, в том числе сложная, на больших объемах данных.
3. Интеграция с форматами DeltaLake – parquet, delta, iceberg. Подготовка данных для ML решений.
5. Обработка JSON форматов.
6. Витрина для BI систем – аналог ClickHouse.

Преимущества АХЕ

- Вместо 3 СУБД – одна и более эффективное использование ресурсов
- Быстро и просто
- Не нужна распределенная система и много ядер Greenplum
- Отчеты, аналитические приложения
- DWH, витрины, отчеты, онлайн аналитика
- DataVault, звезда, снежинка
- Теплые данные
- Снижение ТСО

Технологические преимущества

- В ~ 10-50 раз быстрее
- В ~ 5-10 раз меньше объем данных
- Выбор оптимальной системы хранения для горячих и холодных данных
- Эффективный масштабируемый каталог в виде набора SQL таблиц с хорошо документированной схемой
- Неограниченная масштабируемость – как по вычислениям, так и по системе хранения

Дорожная карта

1. Быстрая SIMD конвертация
PG Heap таблиц на лету в колонки

2. Запросы к аналитическим таблицам
pgpro_metastore без удержания горизонта
транзакции PG

3. Table Access Method поверх таблиц
pgpro_metastore

4. Быстрый параллельный CDC на основе
WAL файлов (без использования слотов
репликации)

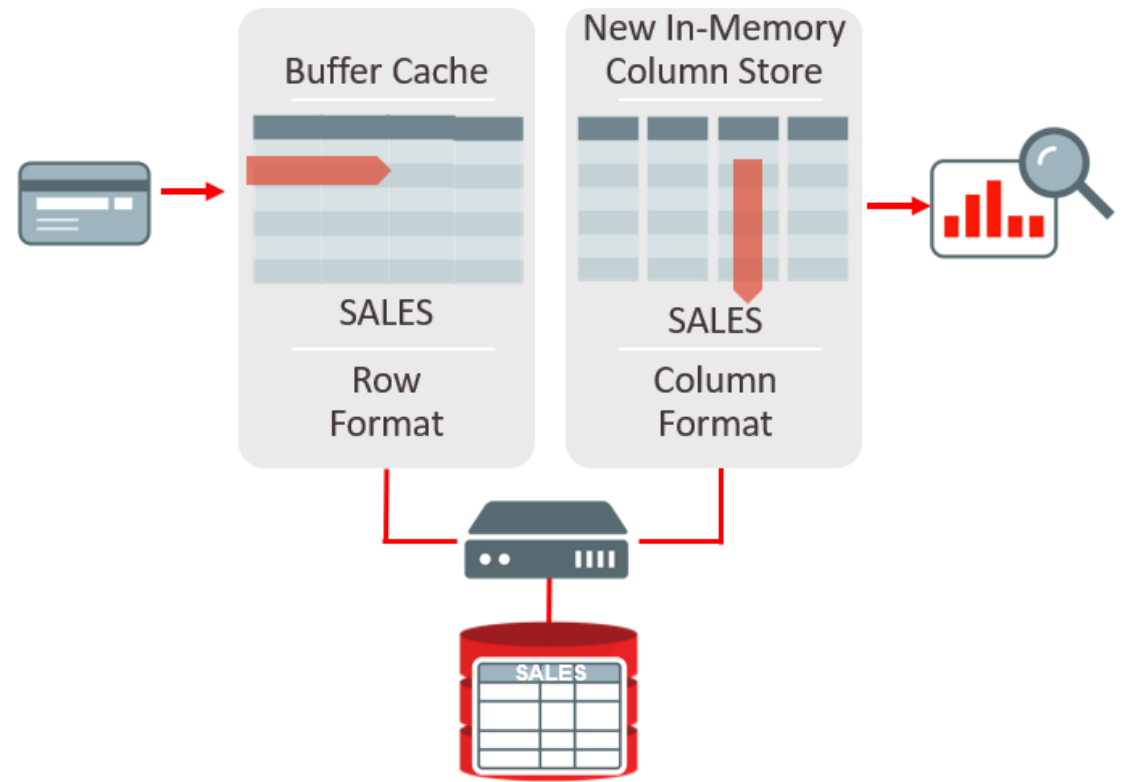
5. Интеграция с продуктами Postgres Pro:
Tengri, Backup, Shardman

HIMERA

(Hybrid In Memory Real Analytics)

In-memory columnar cache

- 2 представления данных таблицы – построчное и поколоночное
- Построчное для OLTP
- Поколоночное для OLAP
- Оптимизатор выбирает оптимальный план
- Обновления на построчном и синхронизация кэшей
- Сильно ускоряет аналитические запросы



HIMERA (Hybrid In-Memory Real Time Analytics)

- Создается командой CREATE INDEXon T1 using himera(c1, c2, c3);
- Ограничение – 32 колонки
- 3 +1 области памяти
- Shared buffer
 - Buffer cache
 - WOPS – изменения копятся но применяются по commit
 - ROPS – меняется асинхронно
- Для отображения всех зафиксированных изменений – в памяти backend локальный ROPS (ROPS+изменения из WOPS) => согласованность
- Оптимизатор учитывает стоимость поколоночных шагов
- Области на диске – для paging, они и WOPS – вспомогательные объекты

Заключение

- OLTP и OLAP решают разные бизнес-задачи
- Правильный выбор архитектуры критичен для производительности
- Современные системы стирают границы между двумя подходами
- Postgres Pro предлагает спектр решений, что позволяет реализовать оптимальную архитектуру системы

Q & A