Реализация Data Vault при построении DWH на PostgreSQL и Greenplum

Проблемы и решения построения высоконагруженного аналитического хранилища данных

Борис Бондарев

Главный архитектор
BI.Qube

BI.Qube

Проблемы построения DWH

Извлечение

С ростом бизнеса увеличивается разнообразие и сложность источников данных

Хранение

Для прогнозирования и адаптации к рыночным изменениям необходимы долгосрочное хранение и историзация

Качество

При всём многообразии пользовательских BI-инструментов требуется подготовка данных

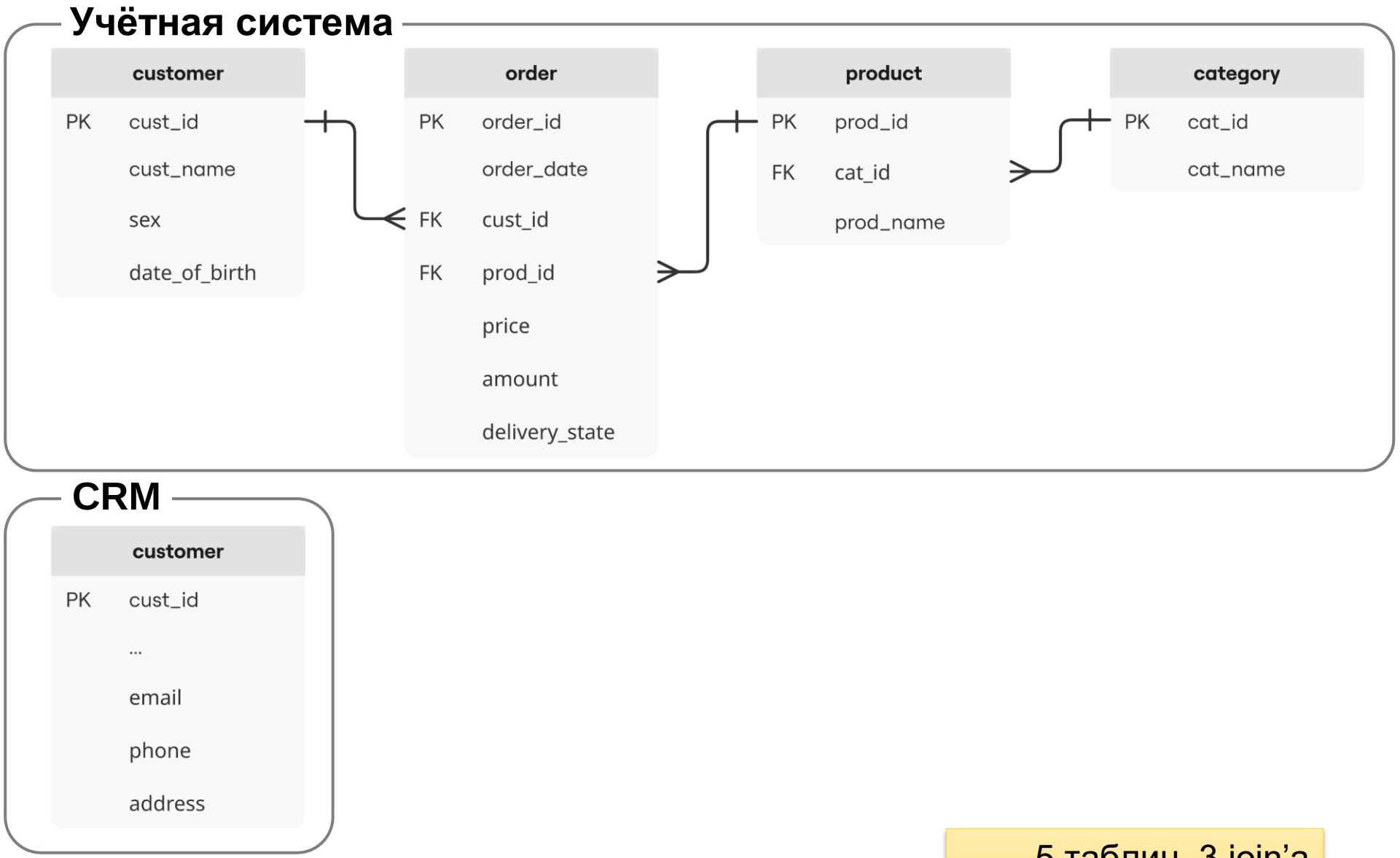
Трудоёмкость и производительность

Требуются глубокие компетенции и значительные трудозатраты для создания оптимального кода

Моделирование

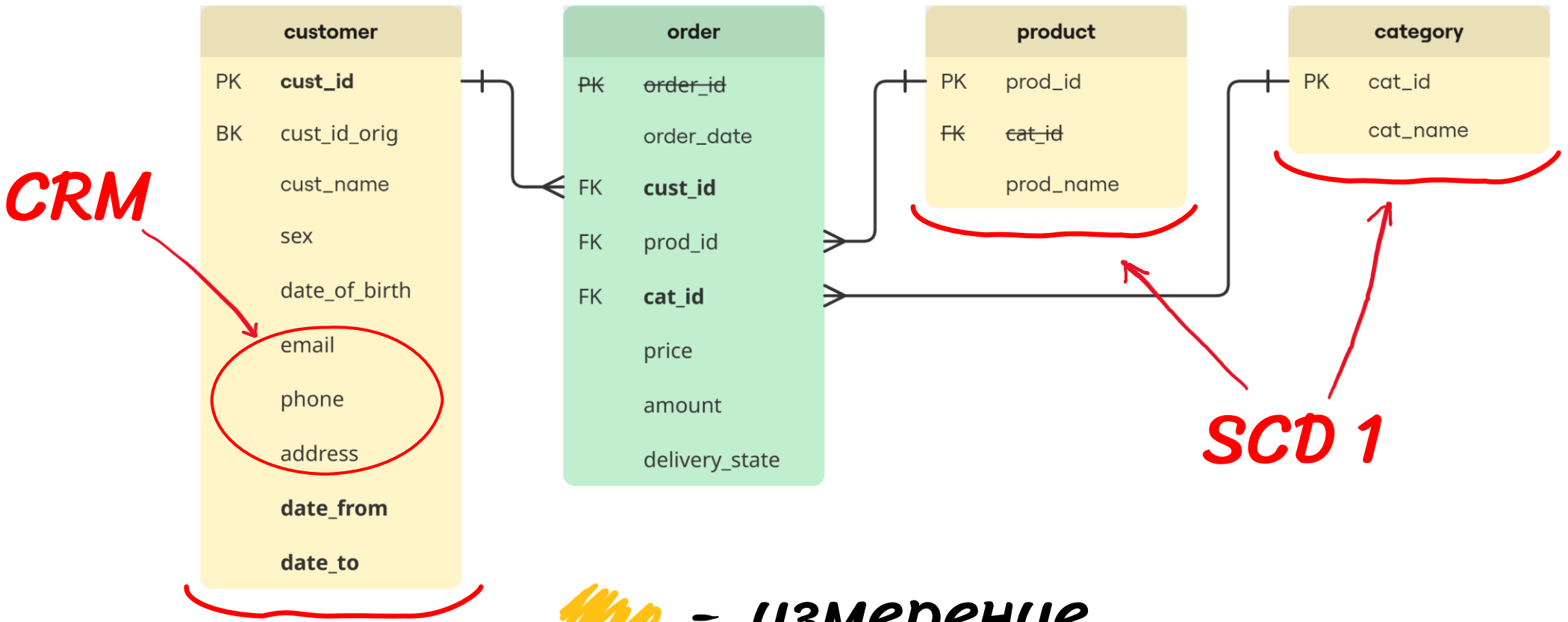
Подготовка модели данных
- самая сложная часть работы с хранилищами данных любого масштаба

“Заказы”. Системы-источники данных



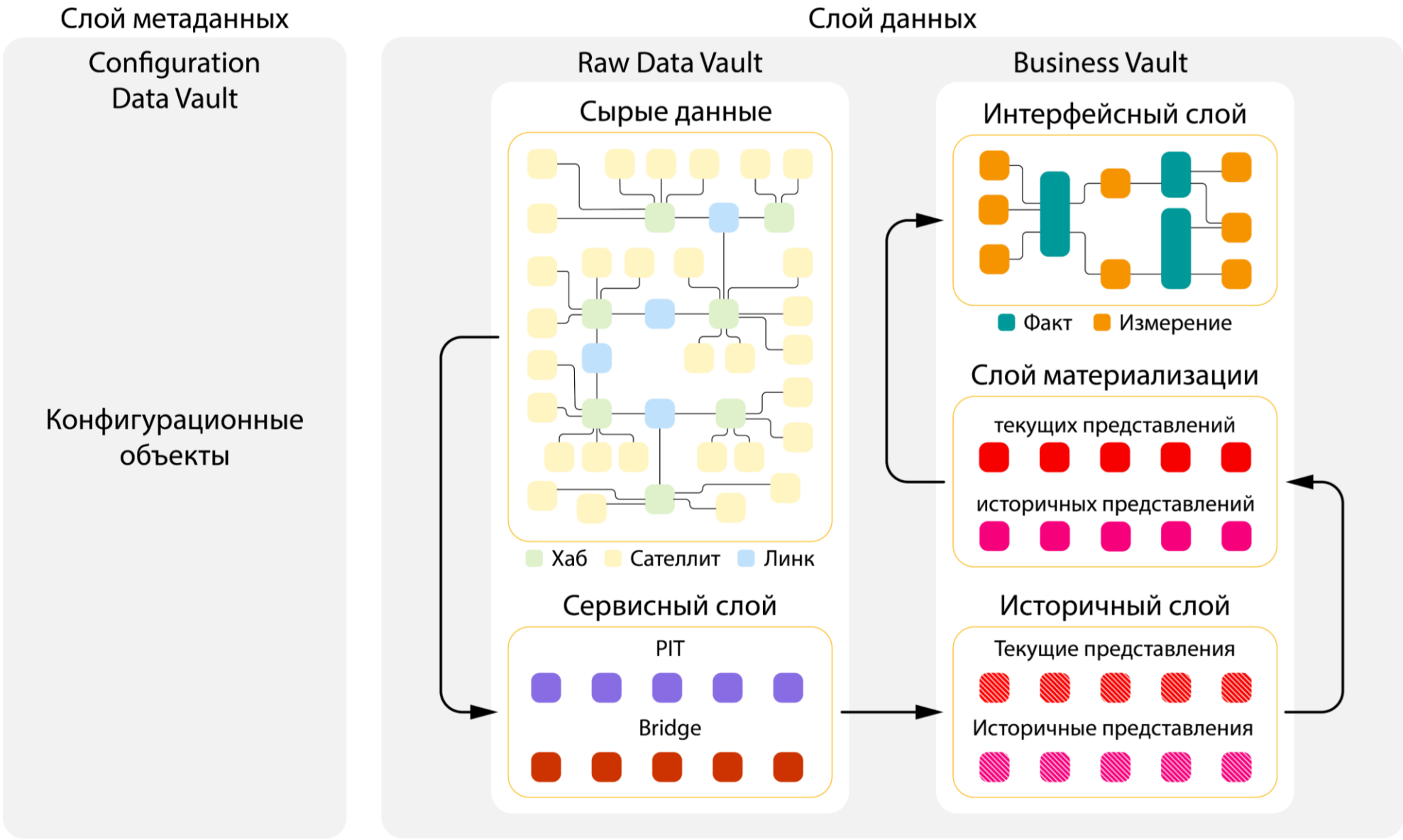
5 таблиц, 3 join’а

“Заказы”. Dimensional model, Star schema



4 таблицы, 3 join’а

Методология Data Vault построения DWH



“Заказы”. Data Vault model. Raw Data Vault

Хабы:

- генерируем суррогатный ключ для каждого бизнес-ключа

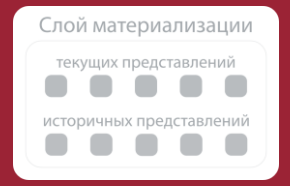
hub_customer	
PK	cust_id
BK	cust_bk

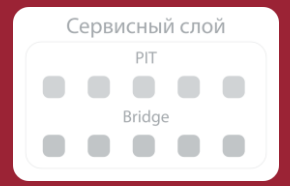
hub_order	
PK	order_id
BK	order_bk

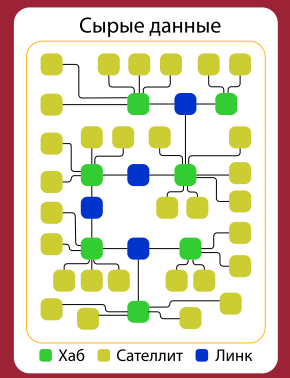
hub_product	
PK	prod_id
BK	prod_bk

hub_category	
PK	cat_id
BK	cat_bk

4 таблицы, 0 join’ов



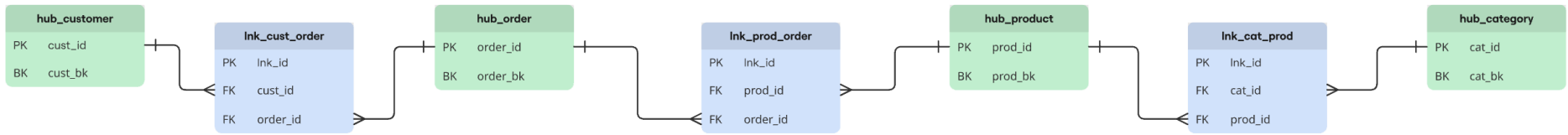




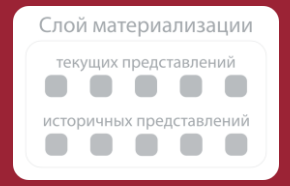
“Заказы”. Data Vault model. Raw Data Vault

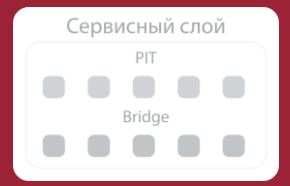
Линки:

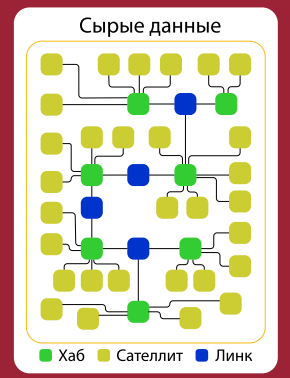
- фиксируем связи



7 таблиц, 6 join’ов



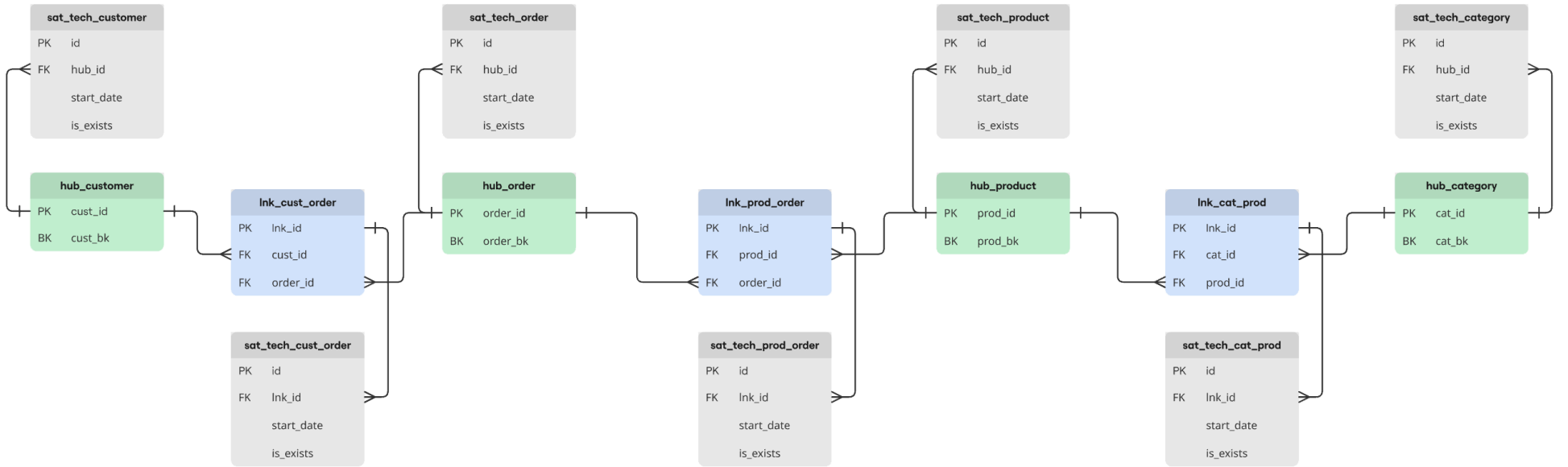




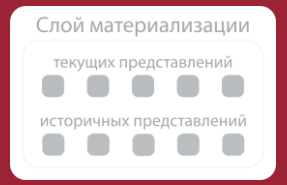
“Заказы”. Data Vault model. Raw Data Vault

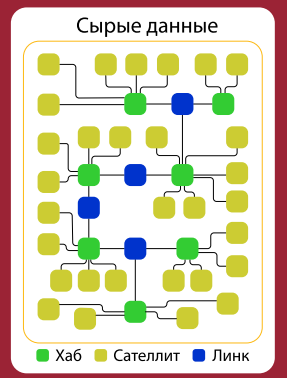
Технические сателлиты:

- историчность появления/удаления хабов/линков



14 таблиц, 13 join’ов

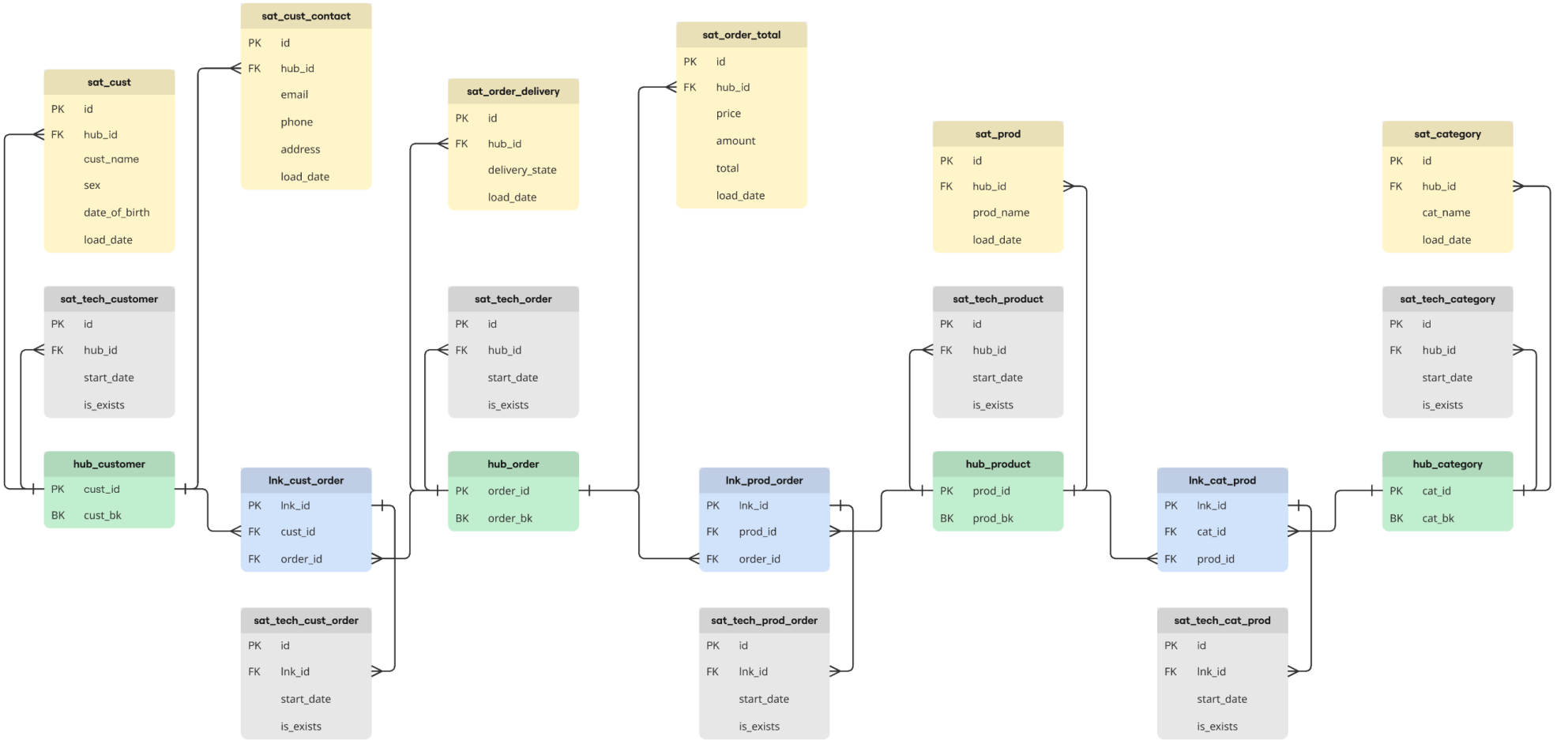




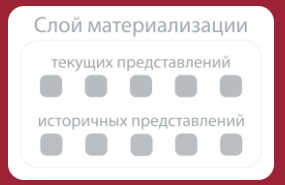
“Заказы”. Data Vault model. Raw Data Vault

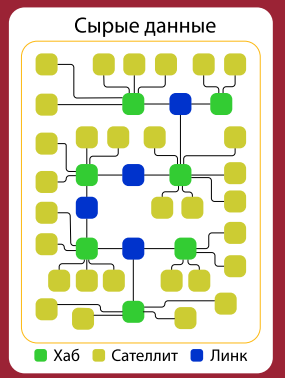
Сателлиты:

- фиксируем значения и историю изменения атрибутов



20 таблиц, 19 join’ов

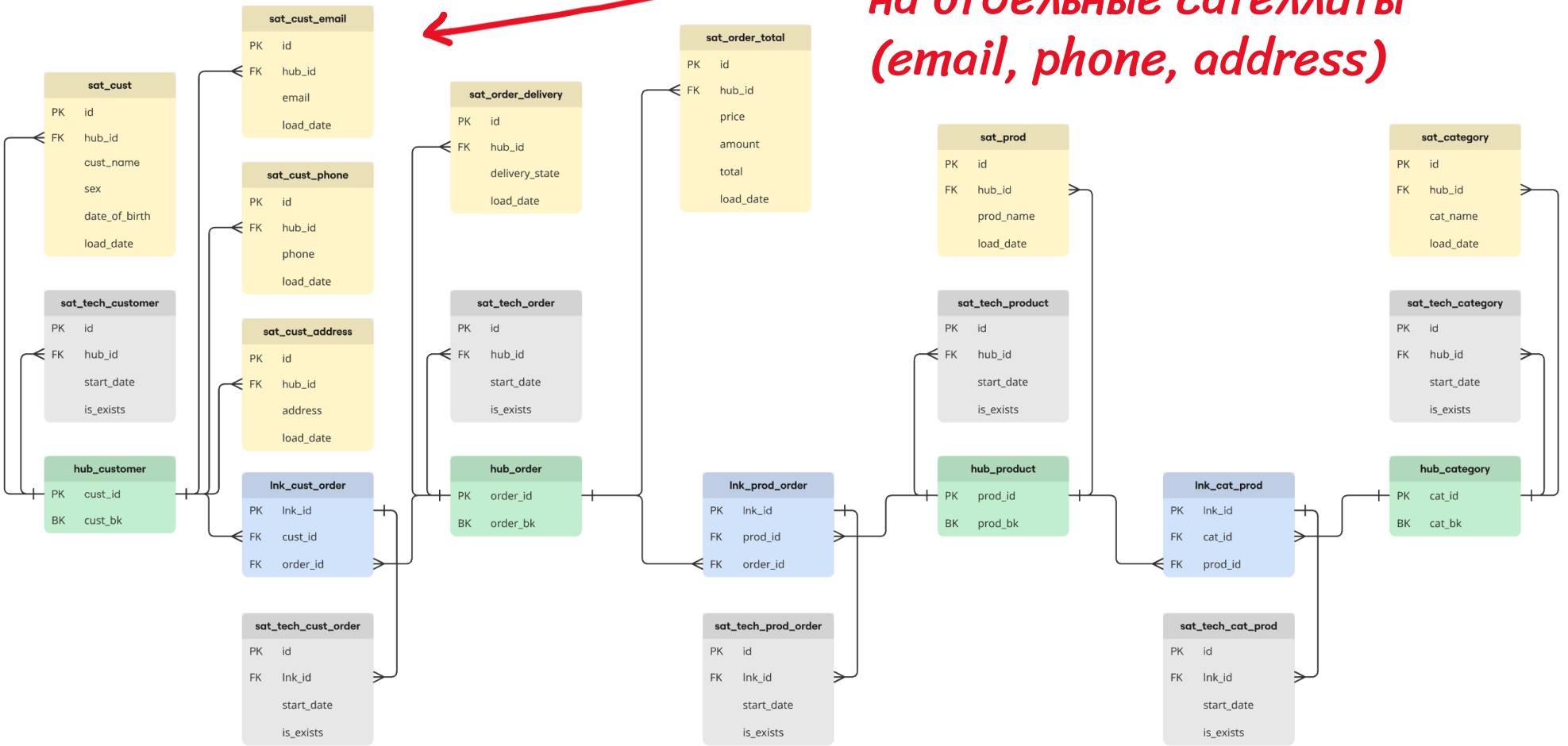




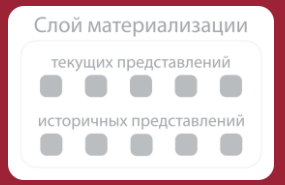
“Заказы”. Data Vault model. Raw Data Vault

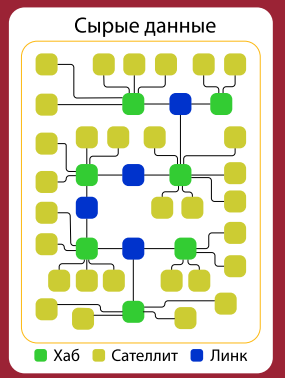
Разбиение на сателлиты производится как по источникам, так и по частотности изменений

Разбили сателлит “контакты” на отдельные сателлиты (email, phone, address)



22 таблицы, 21 join

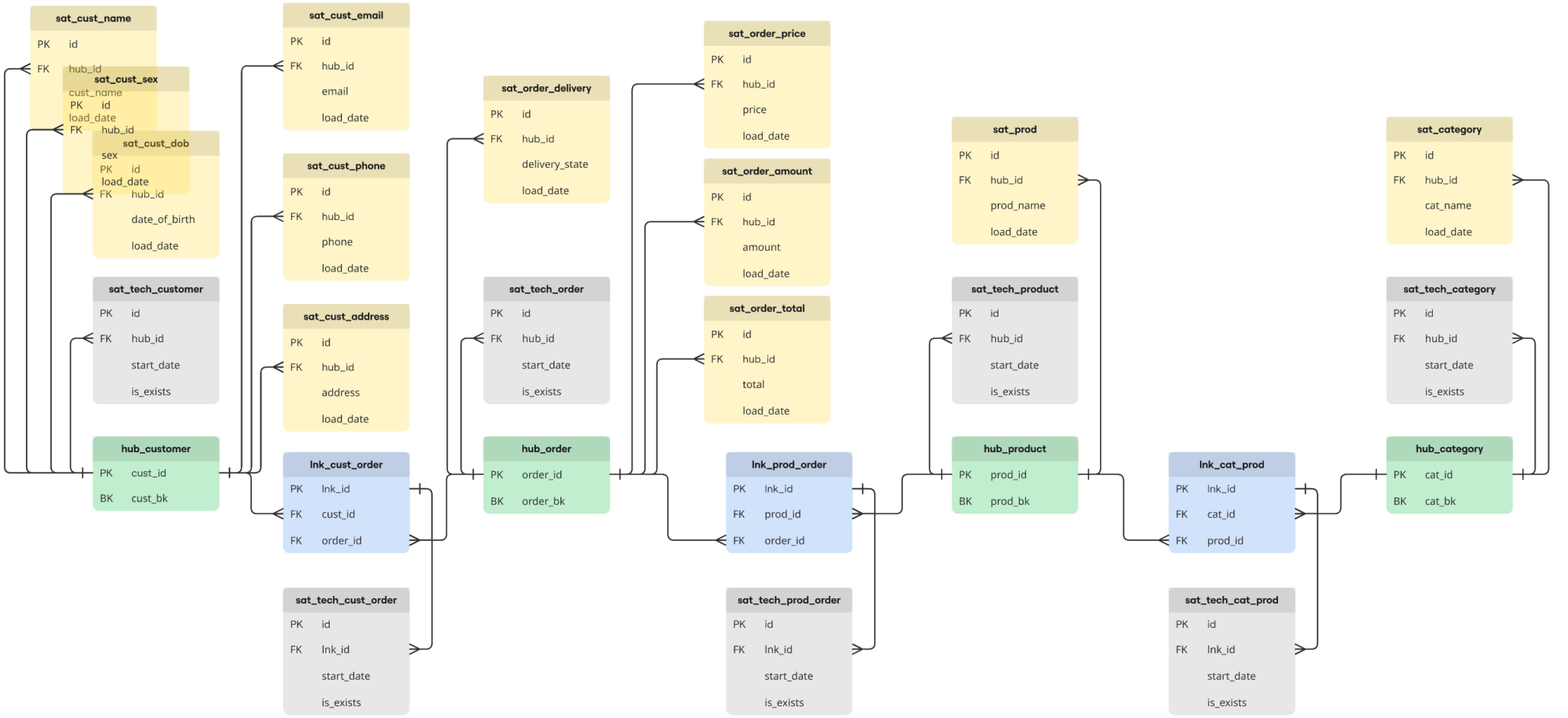




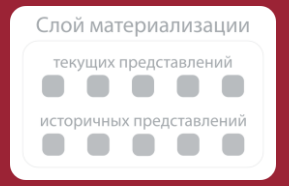
“Заказы”. Data Vault model. Raw Data Vault

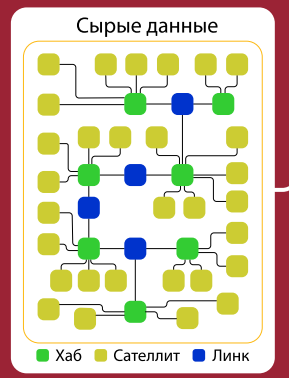
В пределе каждый сателлит может содержать единственный атрибут…

…и мы приходим к Anchor модели!



26 таблиц, 25 join’ов





“Заказы”. Data Vault model. Business Vault

В сервисном слое формируем вспомогательные объекты:
PIT- и Bridge-таблицы (при необходимости)

pit_customer	
PK	id
FK	hub_id
	pit_load_date
	sat_tech_cust_load_date
	sat_cust_load_date
	sat_cast_contact_load_date

pit_order	
PK	id
FK	hub_id
	pit_load_date
	sat_tech_order_load_date
	sat_order_total_load_date
	sat_order_delivery_load_date

pit_product	
PK	id
FK	hub_id
	pit_load_date
	sat_tech_prod_load_date
	sat_prod_load_date

pit_category	
PK	id
FK	hub_id
	pit_load_date
	sat_tech_category_load_date
	sat_category_load_date

brg_cust_order_prod_cat	
PK	id
FK	hub_cust_id
FK	hub_order_id
FK	hub_prod_id
FK	hub_cat_id

31 таблица, 25 join’ов

Таблица Point-In-Time (PIT)

Проблема:
согласование выборок по времени в случае множества сателлитов

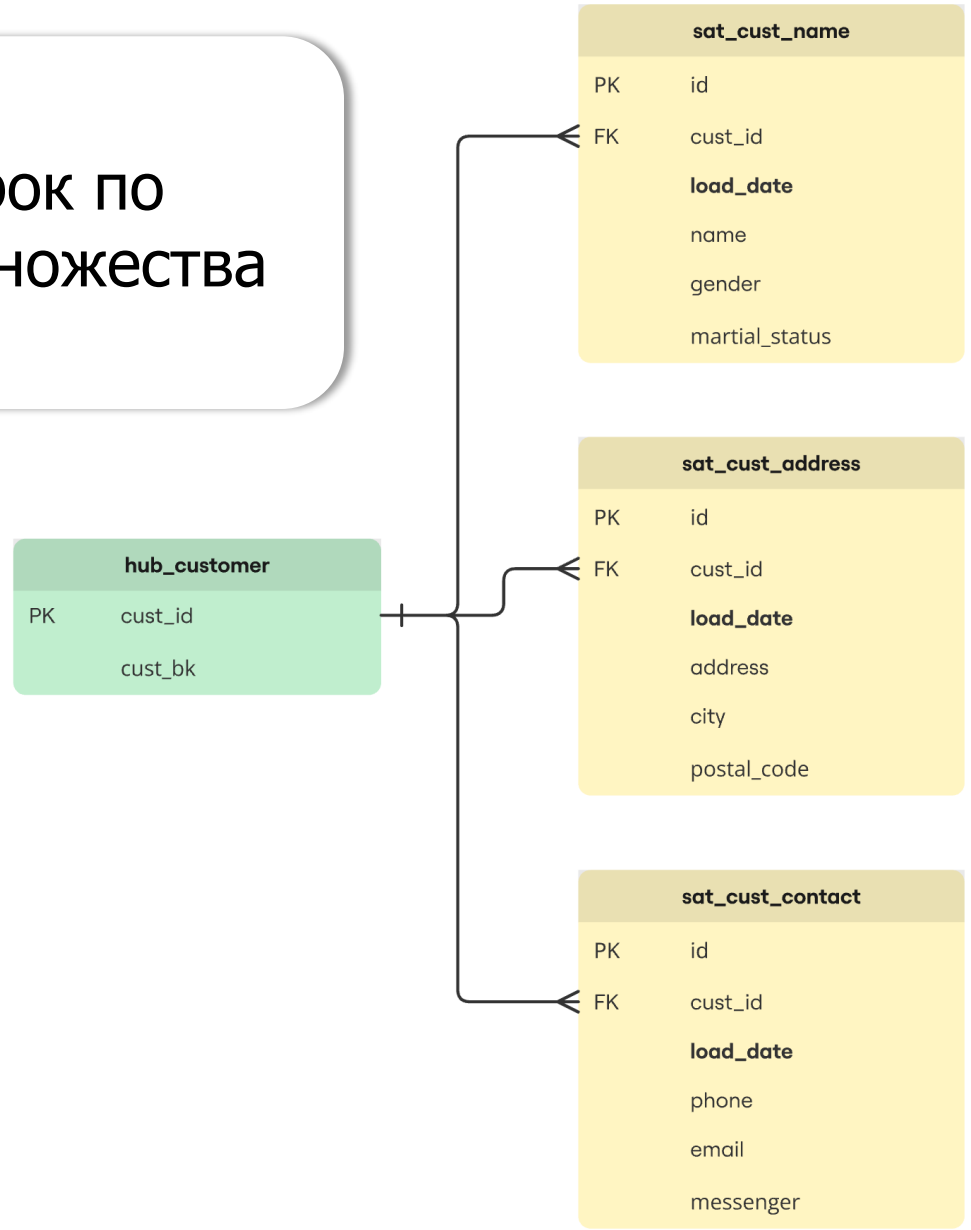


Таблица Point-In-Time (PIT)

PIT
необязательная структура методологии Data Vault 2.0

Поддерживает целостность join’ов по времени для всех сателлитов, относящихся к определённому хабу

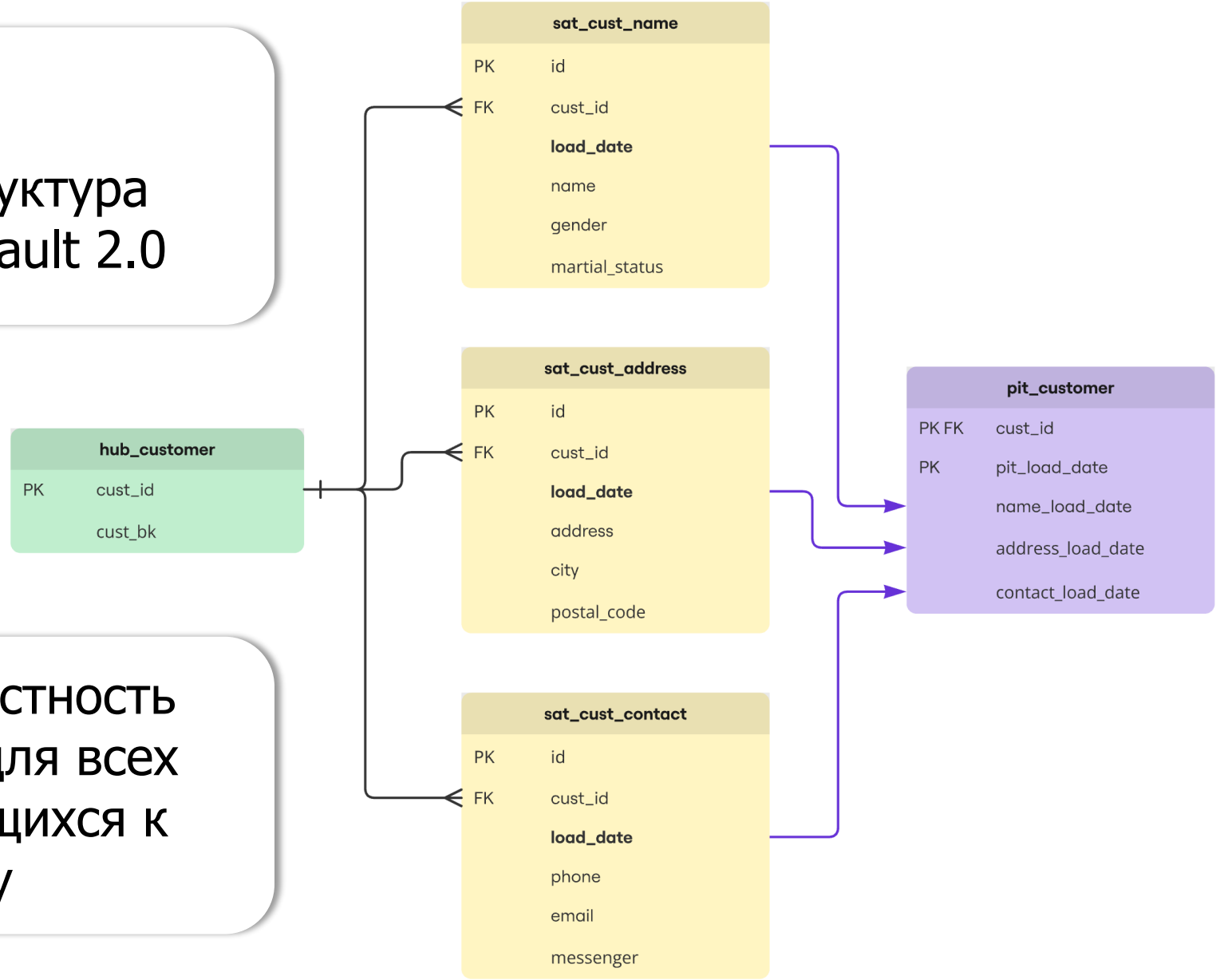


Таблица Point-In-Time (PIT)

sat_name

cust_id	load_date	name	gender	marital_status	record_source
1000	01.07.2024	Иван	мужчина	холост	crm

sat_address

cust_id	load_date	address	city	postal_code	record_source
1000	01.07.2024	ул. Неизвестная, д. 10	Москва	101000	crm
1000	03.07.2024	ул. Тестовая, д. 25	Санкт-Петербург	190000	crm

sat_phone

cust_id	load_date	phone	record_source
1000	02.07.2024	+7-915-123-4567	crm
1000	05.07.2024	+7-926-987-6543	crm

pit_table

cust_id	pit_load_date	name_load_date	address_load_date	phone_load_date
1000	01.07.2024	01.07.2024	01.07.2024	Null
1000	02.07.2024	01.07.2024	01.07.2024	02.07.2024
1000	03.07.2024	01.07.2024	03.07.2024	02.07.2024
1000	04.07.2024	01.07.2024	03.07.2024	02.07.2024
1000	05.07.2024	01.07.2024	03.07.2024	05.07.2024

Unified Time Line

дата изменения соответствующего сателлита

тиражирование

Таблица Bridge

Проблема:
Агрегация на цепочках hub-link-...hub может тормозить

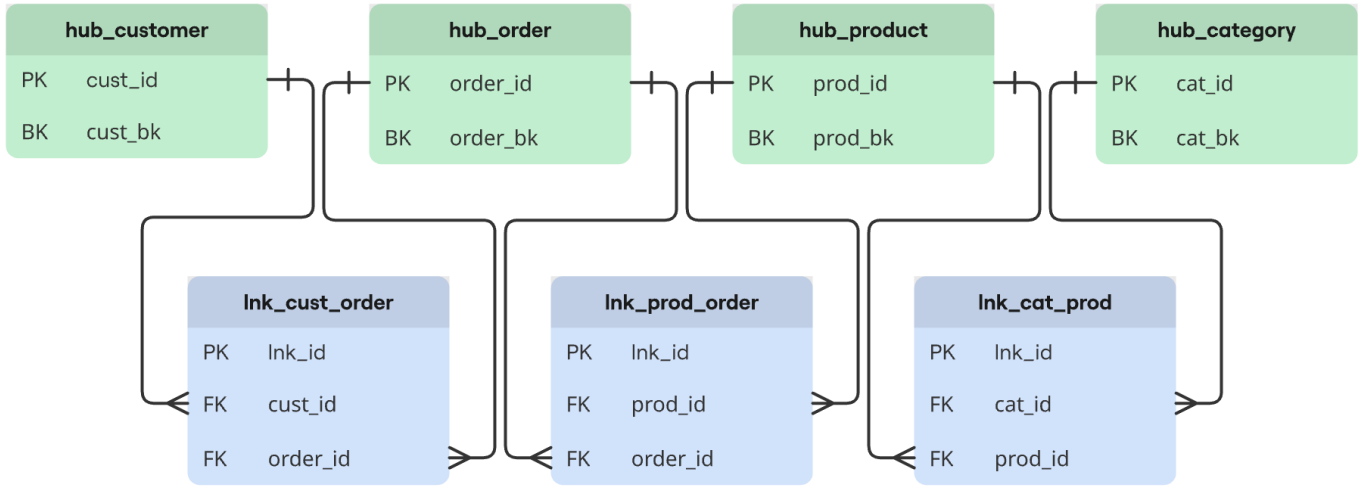


Таблица Bridge

Bridge
необязательная структура методологии Data Vault 2.0

Поддерживает заданную предагрегацию на цепочке hub-link-...-hub для ускорения запросов

“Заказы”. Data Vault model. Business Vault

В историчном слое собираем актуальные значения сущностей и историю

Назначение: скрыть сложность реализации модели (автогенерация SQL) путём предоставления простых интерфейсов, инкапсулирующих логику

Актуальные (на текущий момент времени) данные

act_customer
cust_id
cust_bk
cust_name
sex
date_of_birth
email
phone
address
is_deleted

act_order
order_id
order_bk
order_date
cust_id
prod_id
price
amount
total
delivery_state
is_deleted

act_product
prod_id
prod_bk
cat_id
prod_name
is_deleted

act_category
cat_id
cat_bk
cat_name
is_deleted

Опционально может присутствовать признак удаления записи

Могут быть и отдельные интерфейсы: с удалёнными записями, без них, только удалённые...

31 таблица, 25 join’ов

“Заказы”. Data Vault model. Business Vault

В историчном слое собираем актуальные значения сущностей и историю

Назначение: скрыть сложность реализации модели (автогенерация SQL) путём предоставления простых интерфейсов, инкапсулирующих логику

Актуальные (на произвольный момент времени) данные

act_customer (t)
cust_id
cust_bk
cust_name
sex
date_of_birth
email
phone
address
is_deleted

act_order (t)
order_id
order_bk
order_date
cust_id
prod_id
price
amount
total
delivery_state
is_deleted

act_product (t)
prod_id
prod_bk
cat_id
prod_name
is_deleted

act_category (t)
cat_id
cat_bk
cat_name
is_deleted

31 таблица, 25 join’ов

“Заказы”. Data Vault model. Business Vault

В историчном слое собираем актуальные значения сущностей и историю

Назначение: скрыть сложность реализации модели (автогенерация SQL) путём предоставления простых интерфейсов, инкапсулирующих логику

История изменений в формате SCD, Type2

hst_customer
cust_id
cust_bk
cust_name
sex
date_of_birth
email
phone
address
date_from
date_to

hst_order
order_id
order_bk
order_date
cust_id
prod_id
price
amount
total
delivery_state
date_from
date_to

hst_product
prod_id
prod_bk
cat_id
prod_name
date_from
date_to

hst_category
cat_id
cat_bk
cat_name
date_from
date_to

Временной диапазон актуальности записи

31 таблица, 25 join’ов

“Заказы”. Data Vault model. Business Vault

В **слое материализации** фиксируется состояние произвольного набора сущностей из исторического слоя

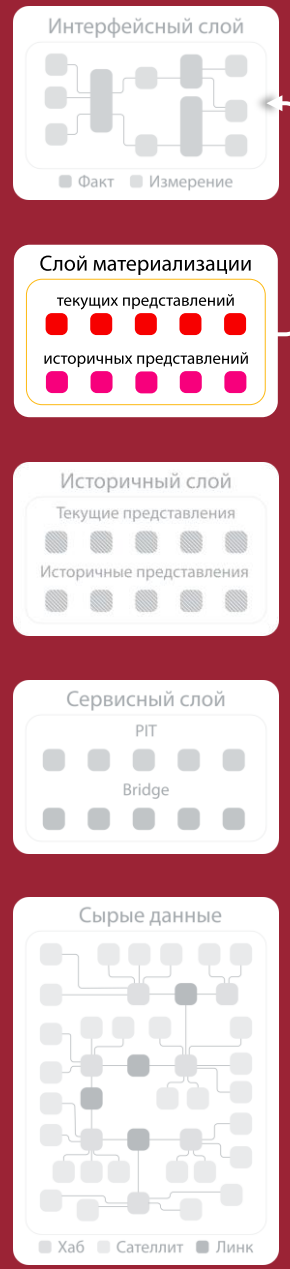
- Использование этого слоя (как и всех прочих, кроме слоя сырых данных) – опционально

Основное назначение – снижение нагрузки к business vault за счёт уменьшения количества join’ов при обращении к материализованным сущностям. Растёт время выполнения ETL и объем требуемого дискового пространства

Актуально для ускорения ad-hoc-аналитики на business vault

Как правило, требует специальных техник обновления материализованных сущностей, т.к. использование функциональности материализованных представлений PostgreSQL и Greenplum приводит к full scan

31 таблица, 25 join’ов

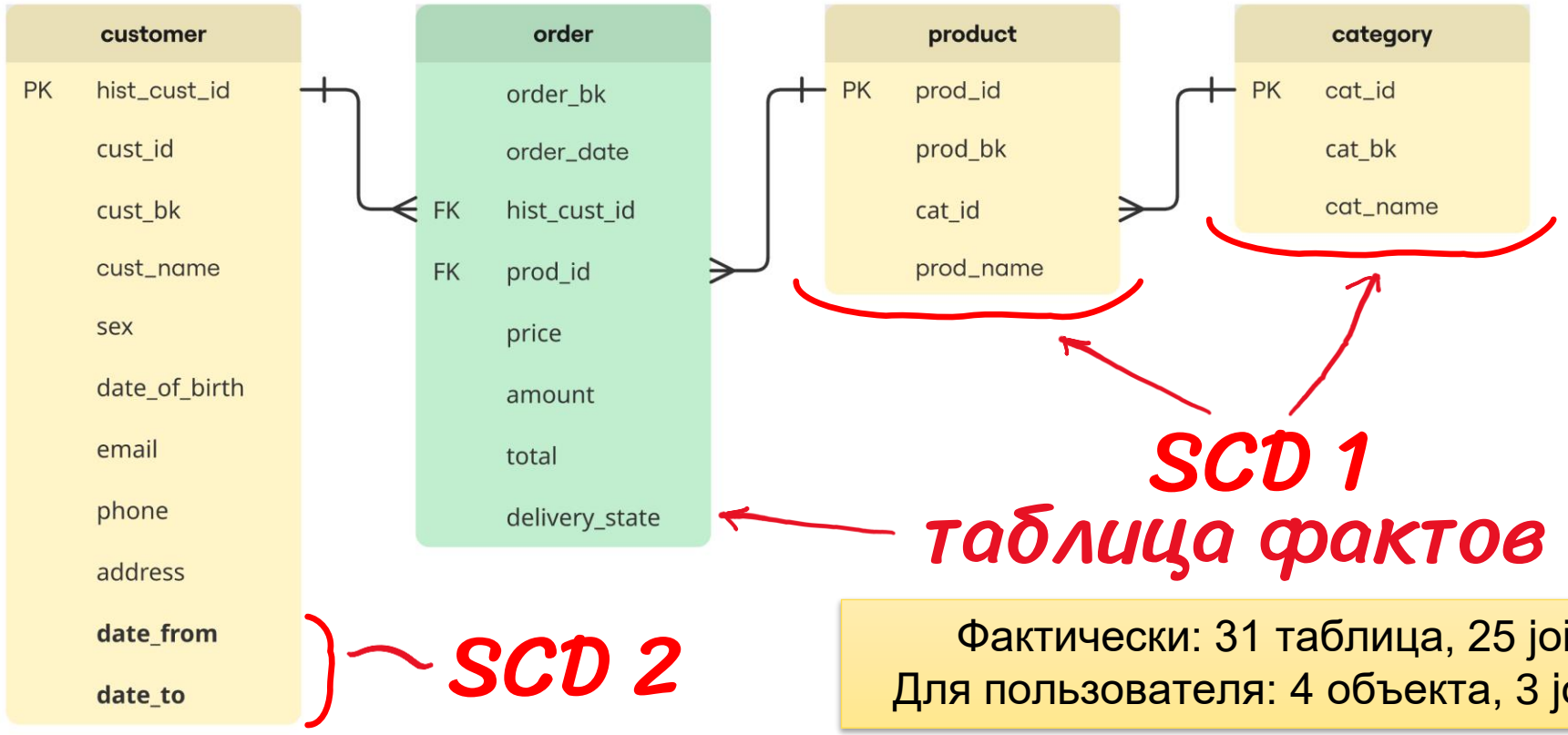


“Заказы”. Data Vault model. Business Vault

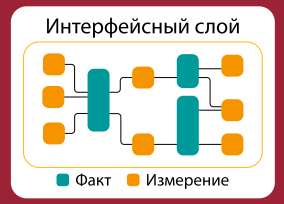
Интерфейсный слой заточен под потребителя и содержит дополнительную бизнес-логику (в т.ч. по вычислениям агрегатов в нужных разрезах и т.д.)

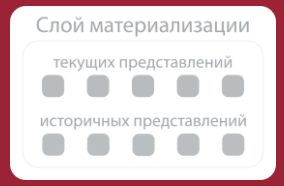
- Большинство пользователей DWH взаимодействуют с витринами или этим слоем, не опускаясь на более нижние

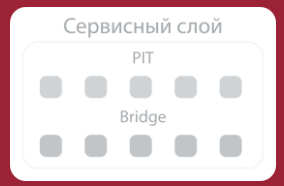
Интерфейсный слой в формате Dimensional model, Snowflake schema

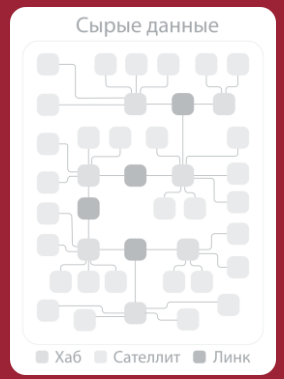


Фактически: 31 таблица, 25 join’ов
Для пользователя: 4 объекта, 3 join’а









Сравнение методологий построения ХД

Характеристика	Dimensional	Data Vault	Anchor
Количество join'ов (зависит от количества)	x1 (измерений)	~x10 (сателлитов)	~x100 (атрибутов)
Производительность	лучшая	средняя	худшая
Разработка	легко	средне	сложно
Сопровождение	сложно	легко	легко
Проектирование	сверху вниз	снизу вверх	снизу вверх
Выборки из модели	просто	зависит от автоматизации шагов трансформации	Сопоставима с Data Vault – больше объектов, но понятийно проще

Преимущества модели Data Vault

Гибкость и расширяемость

Добавление новых сущностей или изменение существующих перестает быть проблемой из-за разложения сущности на составляющие, изменения необходимо вносить только в небольшую часть модели

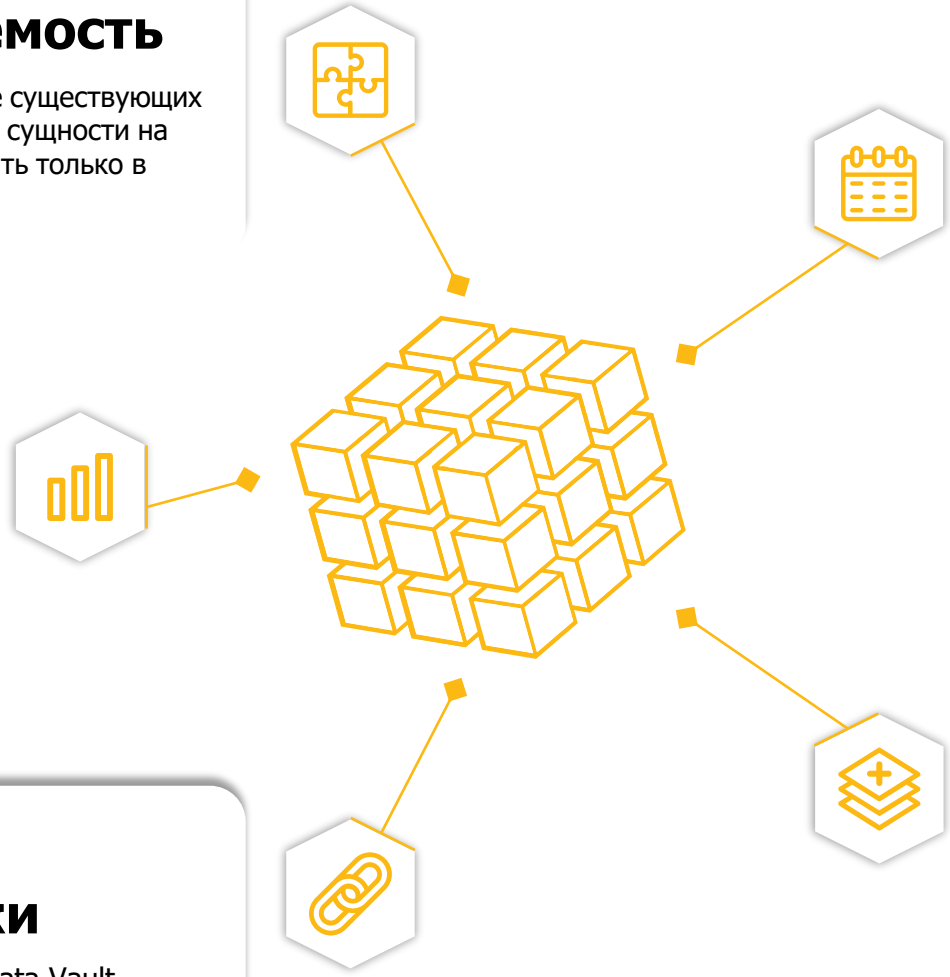
Историзация

Основная функция Data Vault – сохранение полной истории данных. Сущности Satellite используются для хранения всех изменений

В модели создаются и заполняются специальные сущности Technical Satellite, предназначенные для сохранения истории всех добавлений и удалений записи

Инкрементальный процесс обработки

Методология Data Vault позволяет для большинства объектов использовать инкрементальный процесс обработки, что позволяет эффективно использовать вычислительные ресурсы и быстрее выполнять загрузку данных и их обработку



Обновление данных в режиме Insert only

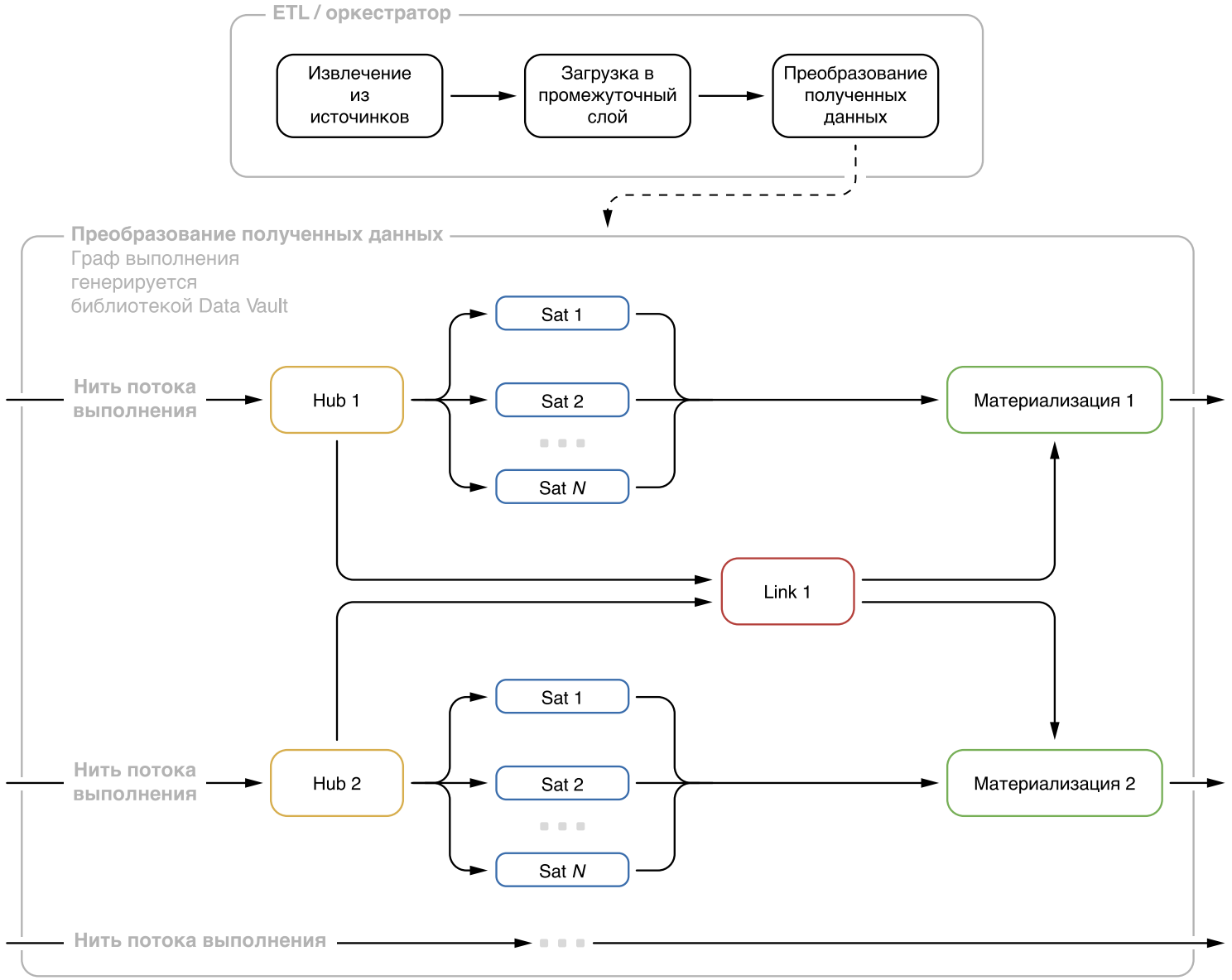
Любые изменения и удаления данных в источнике трансформируются во вставку новых записей в DWH

Не происходит «тяжелых» для DWH операций (обновление и удаление записей), требующих перестроения индексов

Унификация процесса сборки

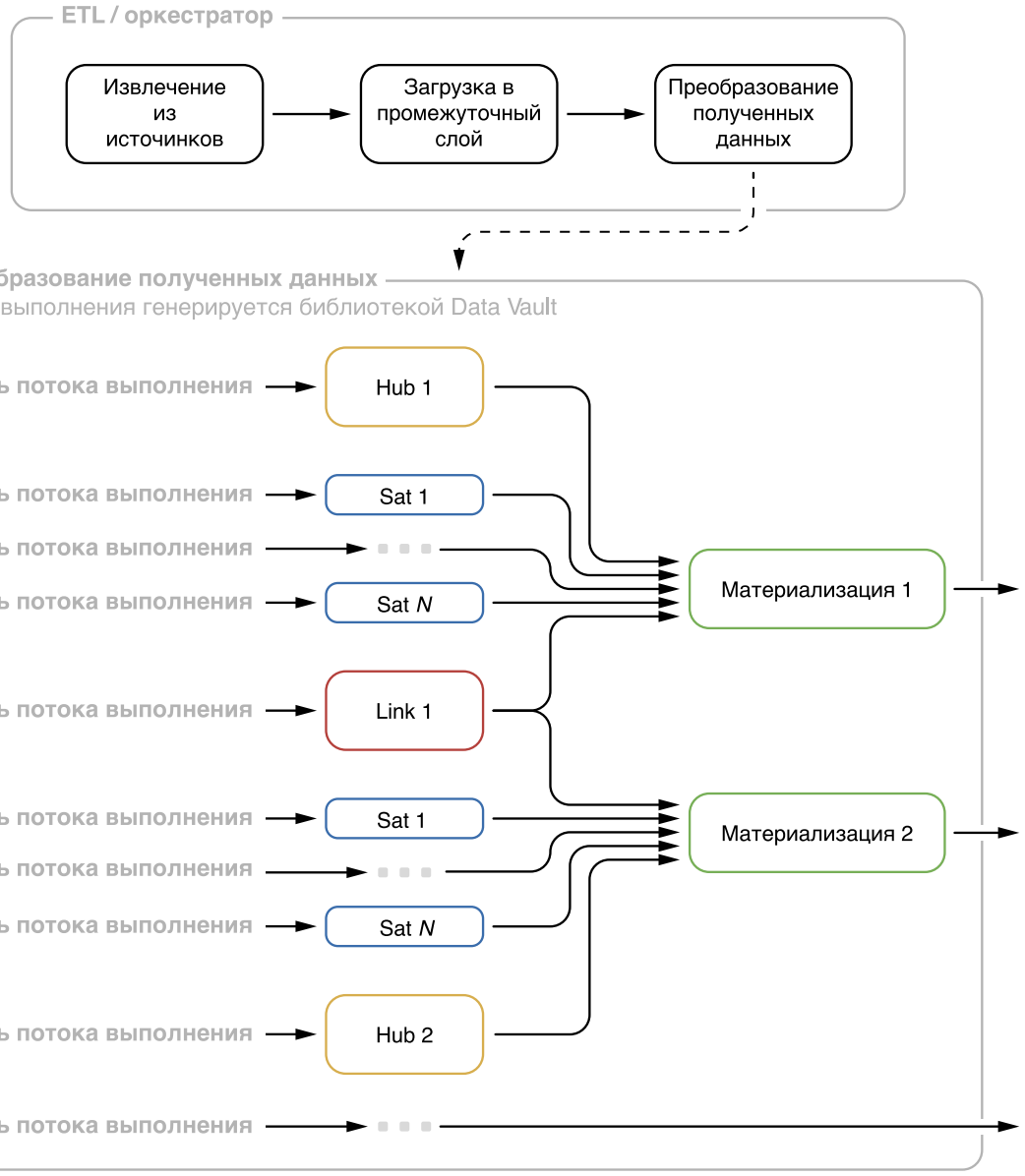
Одинаковые типы сущностей Data Vault различных справочников имеют похожую структуру, позволяющую автоматизировать создание и выполнение ETL-процесса

Параллельный ETL-процесс (ключи на sequence)

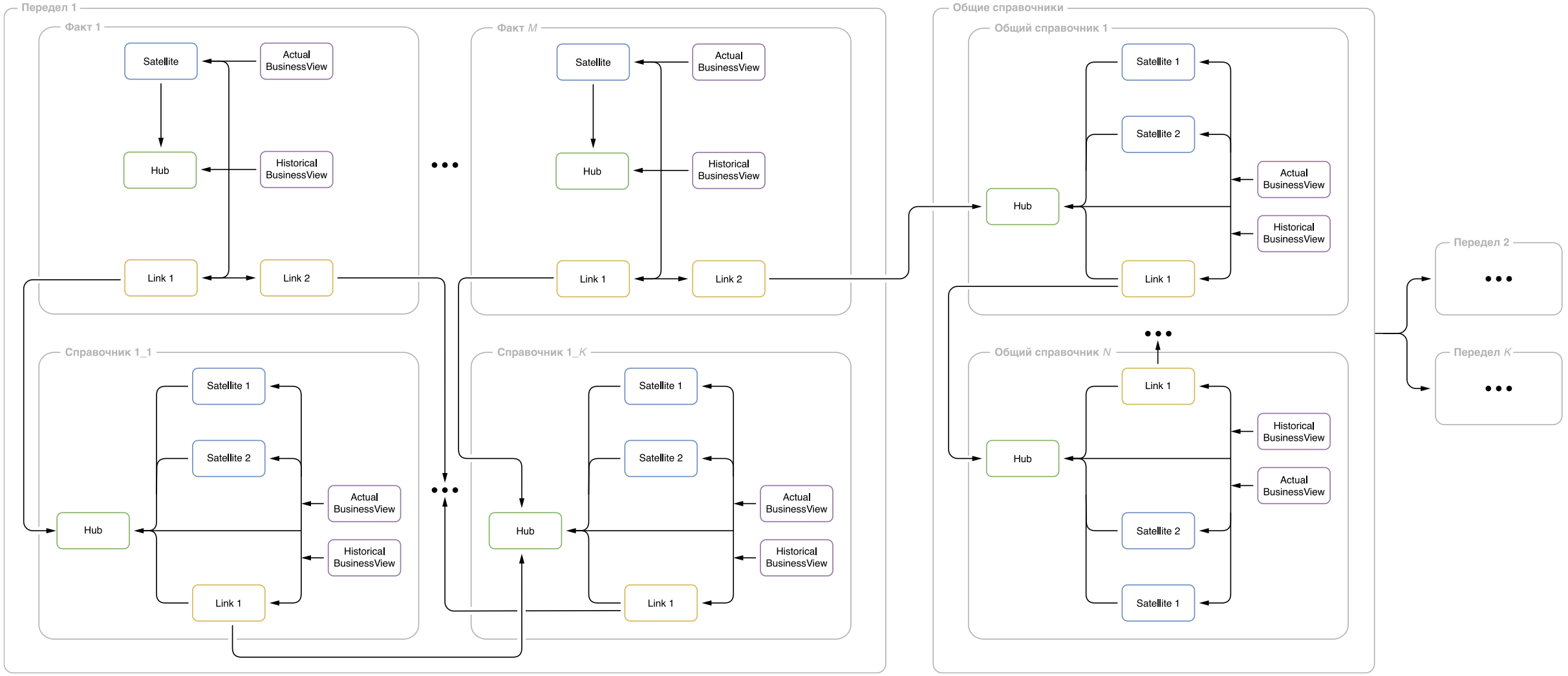


Параллельный ETL-процесс (hash-ключи)

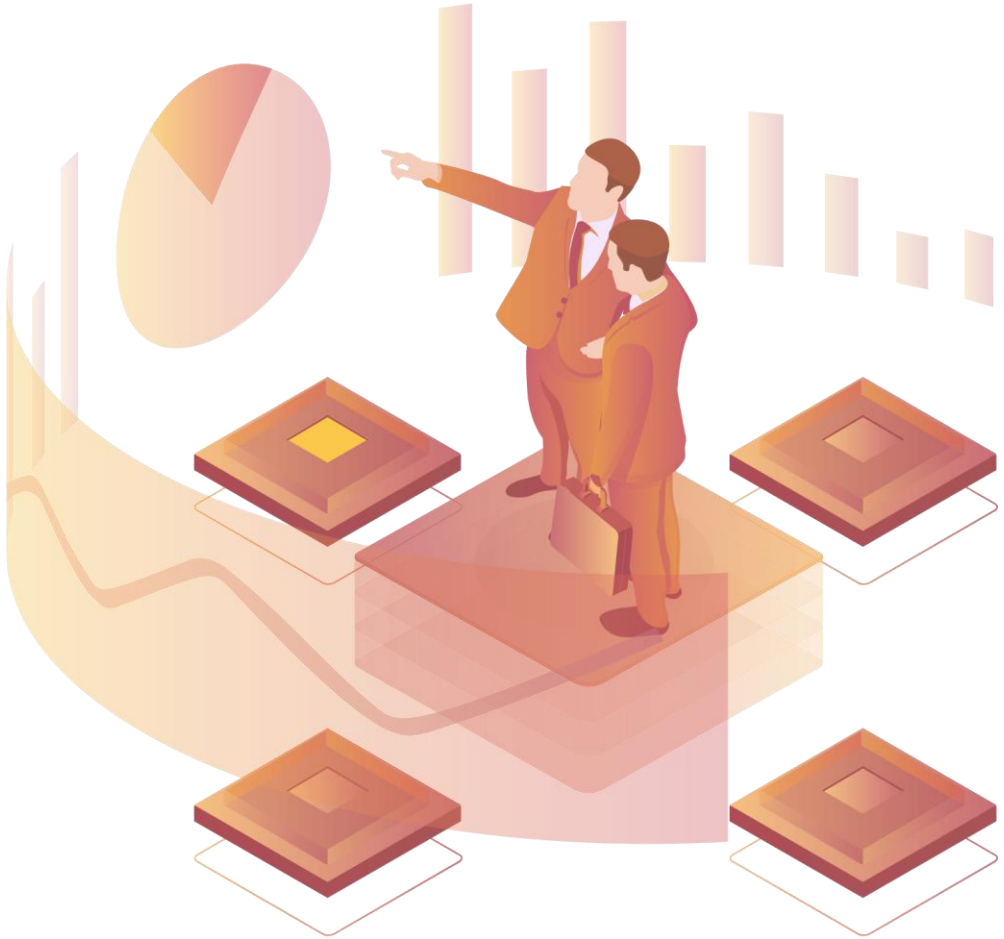
Проблема:
Возможны коллизии



Фрагмент модели Data Vault



Data Vault для эффективной разработки комплексных хранилищ



Эффекты на примере Единой системы производственных показателей (ЕСПП) ЕВРАЗ

BI.Qube

Описание компании заказчика

ЕВРАЗ

- Входит в число крупнейших производителей стали в мире
- Объединяет около 70 000 сотрудников по всему миру
- Выручка российских активов металлургической группы за 2024 год составила $7 190 млн.
- Группа продолжает увеличивать капитальные вложения. CAPEX в 2024 году вырос на 16%, достигнув $736 млн., причем 65% от этой суммы было направлено на проекты развития

Описание проекта

Разработка единого корпоративного хранилища данных производственных показателей

Цели и задачи

- **Построить архитектуру** с использованием только инструментов работы с данными, входящих в реестр РосПО, и open source продуктов
- **Загрузить данные** из систем-источников о производственных показателях и отобразить их на витринах:
  - Разработка модели данных и структуры DWH (Greenplum)
  - Разработка Data Marts (ClickHouse)
  - Разработка ETL-процесса для регулярной загрузки данных (Airflow)
  - Обеспечение Data Lineage
  - Поддержка CI/CD

Постановка задачи для разработчиков DWH

10+
производственных переделов

3 000+
производственных показателей

7:00 LT
доступность обновленных данных

Data Governance
на уровне инструментов

План

1-2 месяца на добавление в DWH следующего передела
~300 показателей

Реальность

Реализация показателей по мере проработки методологами

Работа идёт сразу по всем переделам

Аналитическая проработка

Все показатели делятся на группы

- Плановые и фактические
- Базовые и расчетные

Особенности групп

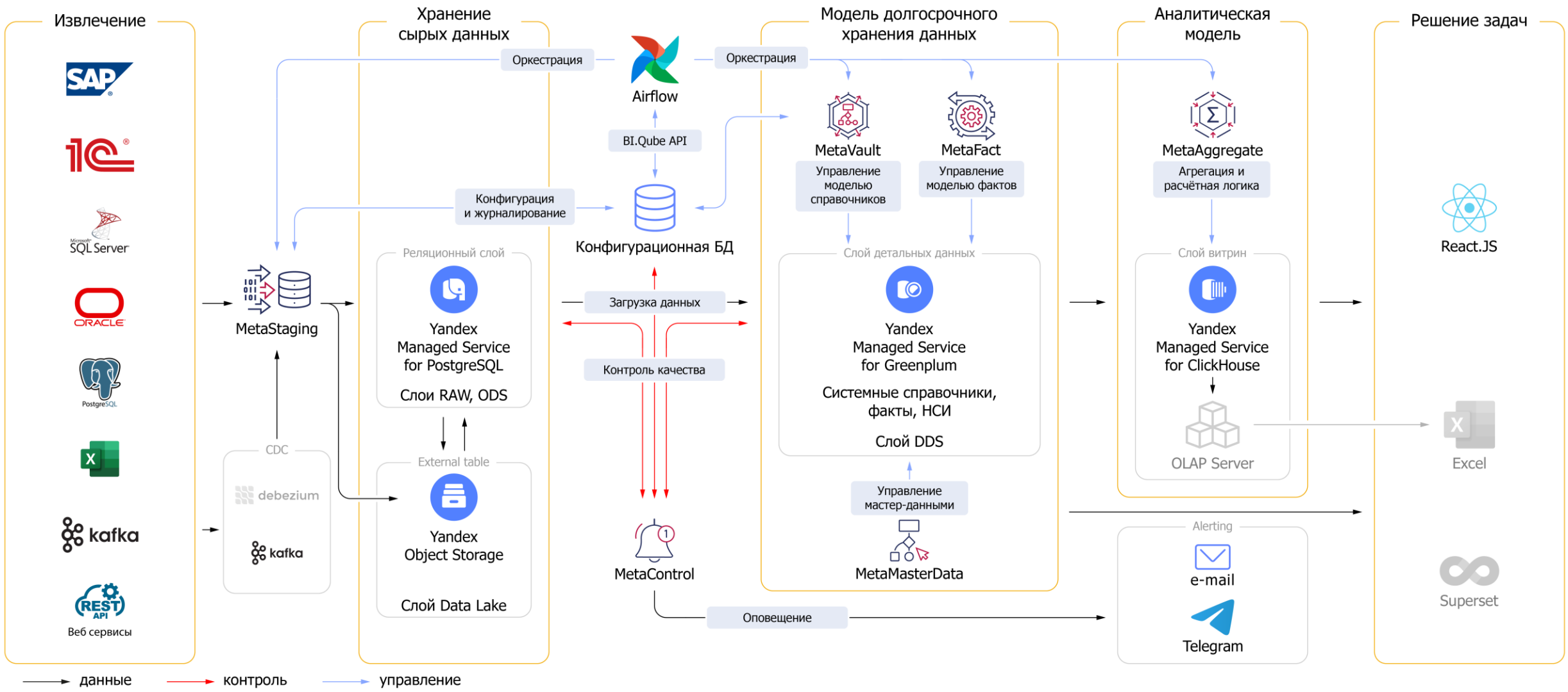
- У каждой группы - свой атрибутный состав, набор справочников и таблиц фактов
- Есть справочники, общие для нескольких групп

Расчет агрегатов на разных уровнях гранулярности

- По времени: смена, день, неделя, месяц, квартал, год
- По производственной структуре: участок, цех, сегмент, предприятие, дивизион
- …

Требуется унифицированный автоматизированный процесс разработки

Архитектура DWH в ЕВРАЗ



Автоматизация процесса разработки

Объект	Автоматизированные операции
Сущности MDM	Объединение справочников с созданием «золотых» записей
Справочники Data Vault	Создание, модификация структуры, актуализация данных и метаданных
Базовые показатели	Обновление таблиц фактов. Работа с инкрементом и секционированием
Агрегаты	Генерация DDL представлений в ClickHouse
Расчетные показатели	Генерация DDL представлений в ClickHouse

Эффект автоматизации загрузки данных

Разработка

Кодирование
вручную

15 минут — полной

60 минут — инкрементальной

120 минут — секционированной

Выполнение

производительность

потребление памяти

Эффект автоматизации загрузки данных

Разработка

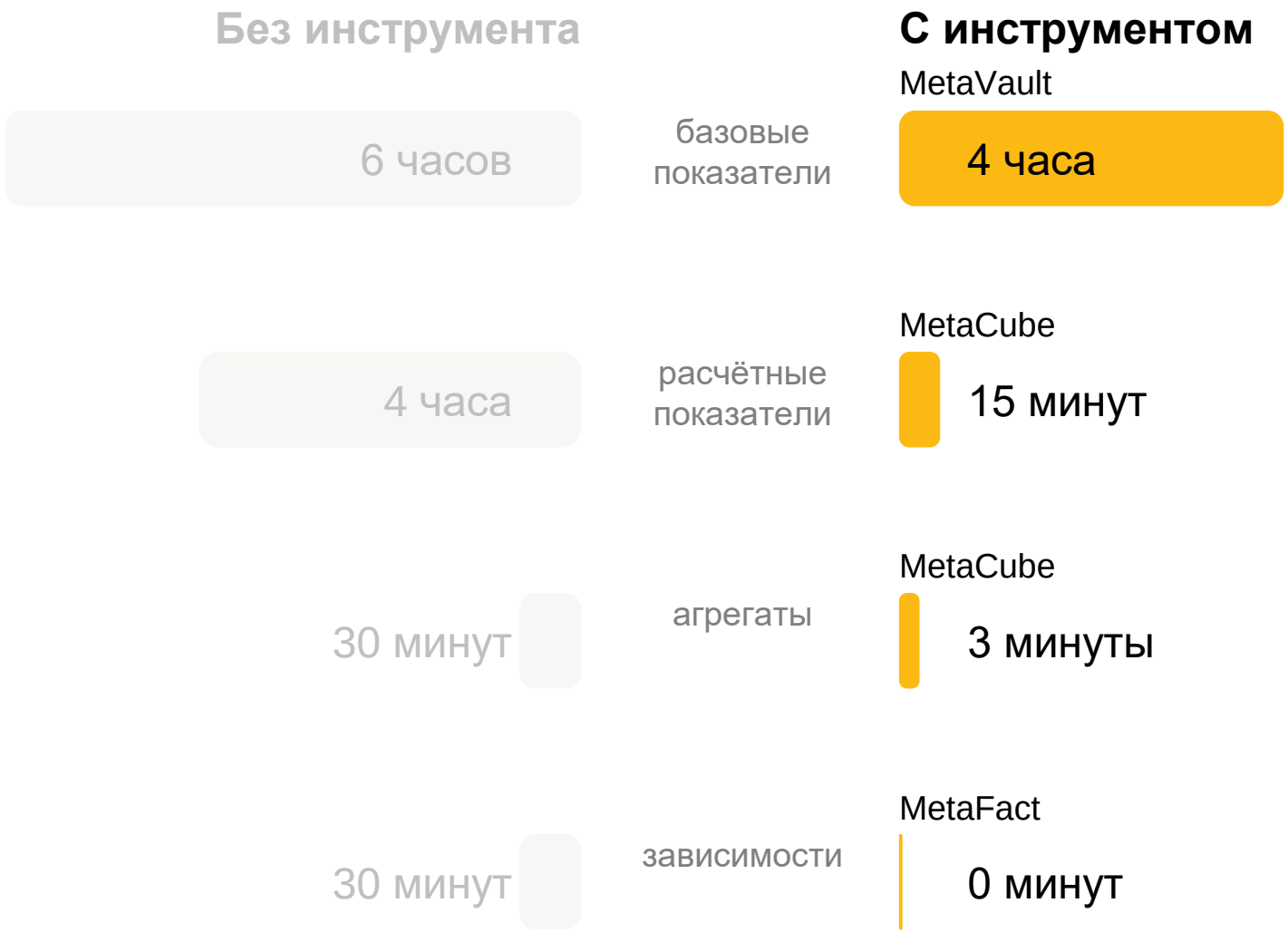
	Кодирование **вручную**	Конфигурирование **в MetaStaging**
полной	15 минут	1 минута
инкрементальной	60 минут	1 минута
секционированной	120 минут	5 минут

Выполнение

производительность	в 2 раза **быстрее**
потребление памяти	в 10 раз **меньше**

Эффект автоматизации разработки показателей

Эффект автоматизации разработки показателей



Эффекты фреймворка в кейсе ЕВРАЗ ЕСПП

10 переделов производства
3 500 показателей

Средняя ставка
30 000 руб. за чел.-день

Вид работы	%	Планируемое количество	Трудоемкость при полностью ручном подходе (чел.-час.)	Итого (чел.-час.)	Трудоемкость с использованием BI.Qube (чел.-час.)	Итого (чел.-час.)
Разработка базовых показателей	70%	2450	6	14700	4	9800
Разработка расчетных показателей	30%	1050	4	4200	0,25	262,5
Корректировка базовых показателей	33%	817	3	2450	2	1633
Корректировка расчетных показателей	33%	350	2	700	0,125	44
Создание агрегатов	100%	3500	0,5	1750	0,05	175
Фиксация зависимостей для построения Data Lineage	100%	3500	0,5	1750	0	0
			Общая трудоемкость (чел.-лет)	**11**		**5**

Средняя стоимость работ
96 млн. руб.

Реализация с BI.Qube
50 млн.руб.

Экономия
46 млн. руб.

Срок проекта
1 год

Снижение требований к команде разработки

Типовая команда на проект из предыдущего кейса

	Microsoft	open source	open source + BI.Qube
Архитектор DWH	MS DWH	open source	open source
Сисадмин, DevOps-инженер	Git, Ansible	Git, Ansible, K8S	Git, Ansible, K8S
Разработчик DWH	T-SQL	PL/pgSQL	PL/pgSQL
Разработчик ETL	MS SSIS, SQL	SQL, Python	SQL, Python
Разработчик витрин	MS SSAS, MDX, DAX	SQL, ClickHouse	SQL, ClickHouse
Аналитик	MS Power BI	SQL, визуализация	визуализация

Проблемы и решения

Выявленные нюансы работы с PostgreSQL и Greenplum

BI.Qube

Трудности практической реализации

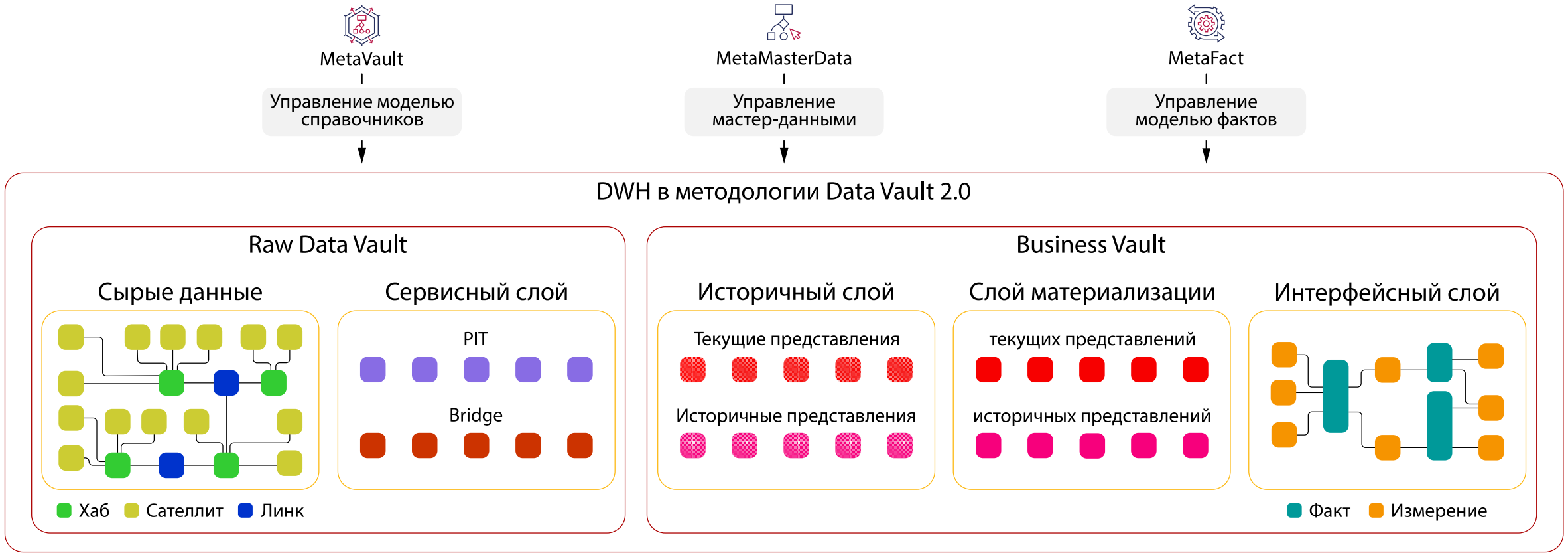
комментарий к статье https://habr.com/ru/articles/504214/

am-habr
1 июн 2020 в 00:07

Хотелось бы узнать о контексте или области хранилища. Потому что мой опыт в телекоммуникации был скорее отрицательным относительно data vault.

Бизнес data vault часто является уровнем для анализа бизнес процессов или поиска ошибок в них. Даже PIT и BRIDGE таблицы не сильно упрощают запрос, если его пишет бизнес-аналитик, а не разработчик. Эти объекты призваны решить техническую проблему со связями n:m
Для восстановления бизнес логики понадобится гигантский запрос со сложными условиями по датам, что добавляет вероятности ошибок.

Слои Data Vault – Raw & Business



Кейс 1. Получение актуального состояния

Задача

- Получить актуальное состояние атрибутов сущности

Применение

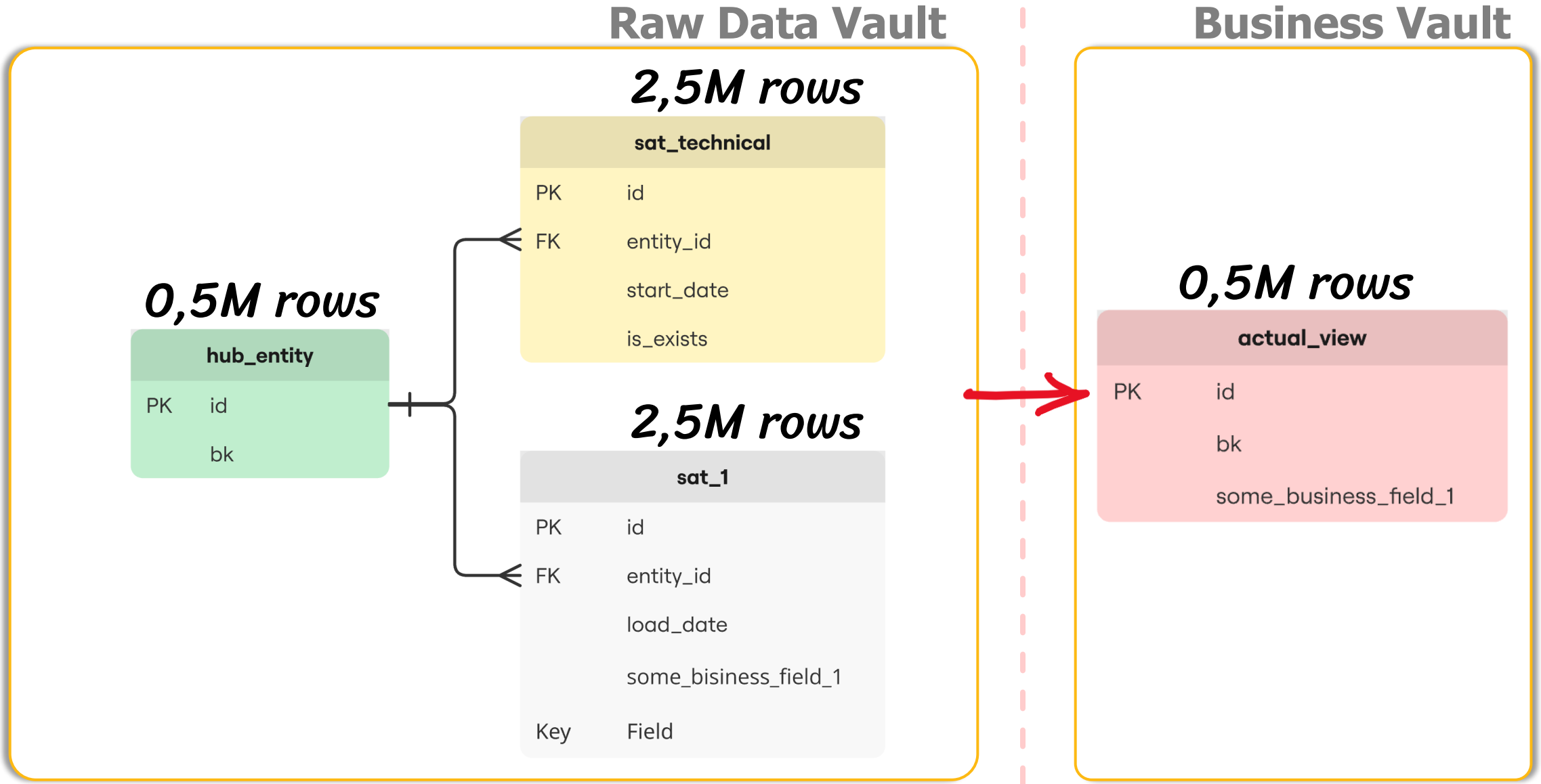
- В процедурах материализации актуального состояния
- В представлениях актуального состояния слоя business-vault
- В процедурах актуализации таблиц слоя raw-data vault

Варианты реализации

1. Доп. атрибут, обновляемый при накате изменений
2. Подзапрос с логикой определения последней записи для каждого идентификатора сущности
3. Использовать PIT-таблицу

Кейс 1. Получение актуального состояния

Модельная задача



Кейс 1. Получение актуального состояния

DDL модели

```
CREATE UNIQUE INDEX
hub_entity_st_ent_date
   ON vault."hub_entity_Sat_Technical"
   (entity_id, start_date)
   include
   (is_exists);


CREATE UNIQUE INDEX
sat_entity_1_ent_date
   ON vault.sat_entity_1
   (entity_id, load_date);
```

```
create table vault.hub_entity
(
   id serial,
   bk int
);
create table vault."hub_entity_Sat_Technical"
(
   id serial,
   entity_id int,
   start_date timestamp,
   is_exists bool
);
create table vault.sat_entity_1
(
   id serial,
   entity_id int,
   load_date timestamp,
   f1 text,
);
```

Кейс 1. Получение актуального состояния

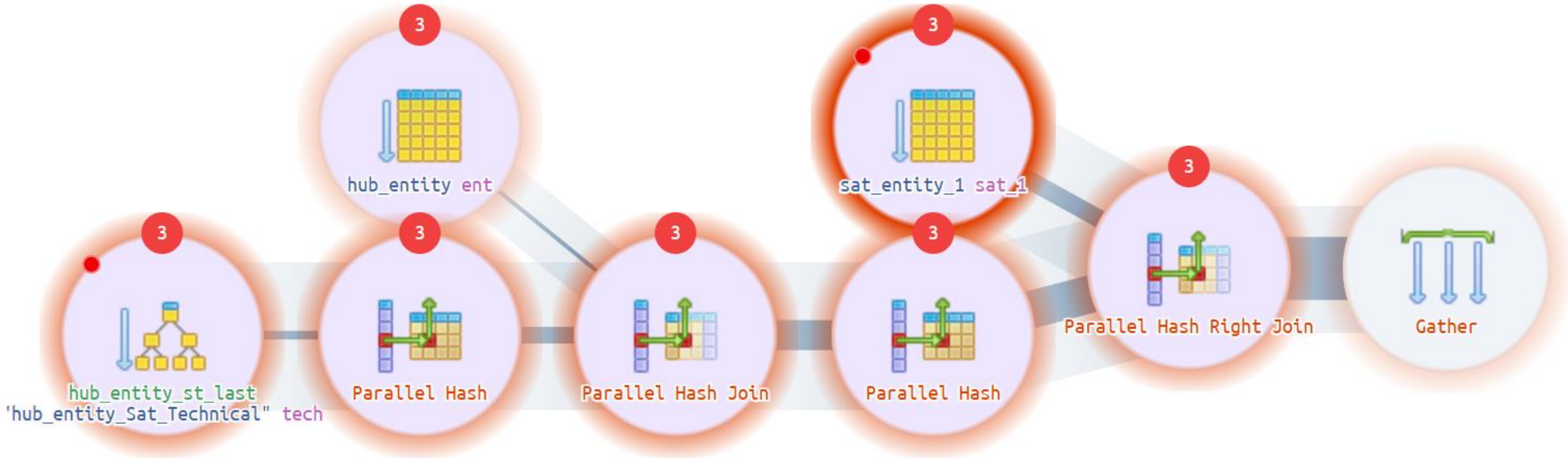
Вариант №1. С поддержкой признака последней записи (PostgreSQL):

```
CREATE UNIQUE INDEX hub_entity_st_last
   on vault."hub_entity_Sat_Technical"
   (entity_id, start_date)
   include
   (is_exists)
   where
   is_last_record;

CREATE UNIQUE INDEX sat_entity_1_last
   ON vault.sat_entity_1
   (entity_id, load_date)
   where
   is_last_record;
```

```
select
   ent.id
   ,ent.bk
   ,sat_1.f1
from
   vault.hub_entity ent

   inner join vault."hub_entity_Sat_Technical" tech on
     tech.entity_id = ent.id
      and tech.is_exists = true
      and tech.is_last_record = true

   left join vault.sat_entity_1 sat_1 on
     sat_1.entity_id = ent.id
      and sat_1.is_last_record = true
```

Кейс 1. Получение актуального состояния

Вариант №1. С поддержкой признака последней записи (PostgreSQL):



Кейс 1. Получение актуального состояния

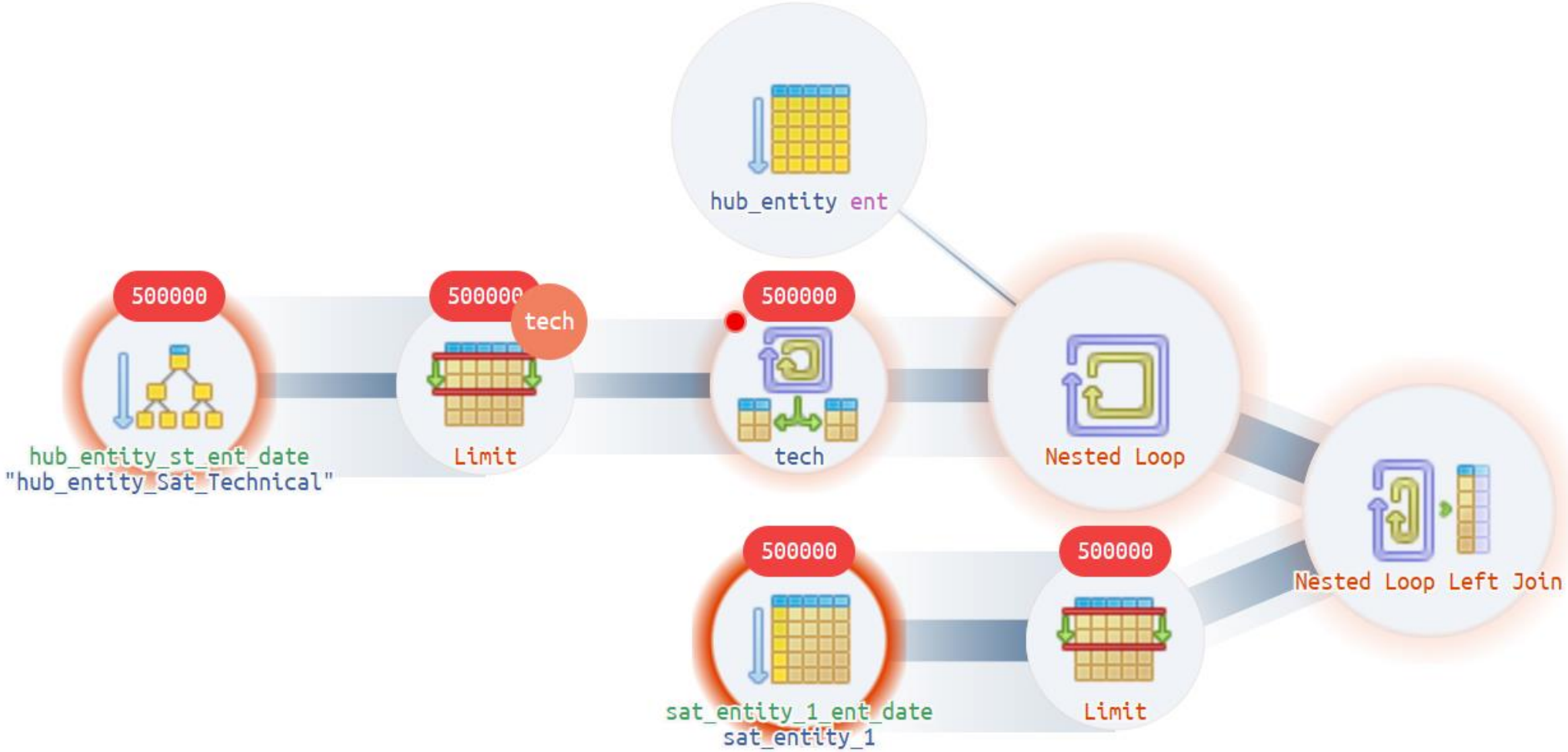
Вариант №2. С подзапросом (PostgreSQL):

```
left join lateral
  (
    select
      *
    from
      vault.sat_entity_1
    where
      entity_id = ent.id
    order by
      load_date desc
    limit 1
  ) sat_1 on true
;
```

```
select
  ent.id
  ,ent.bk
  ,sat_1.f1
from
  vault.hub_entity ent

  inner join lateral
  (
    select
      is_exists
    from
      vault."hub_entity_Sat_Technical"
    where
      entity_id = ent.id
    order by
      start_date desc
    limit 1
  ) tech on tech.is_exists

  left join lateral
  (
    select
      *
    from
      vault.sat_entity_1
    where
      entity_id = ent.id
    order by
      load_date desc
    limit 1
  ) sat_1 on true
;
```

Кейс 1. Получение актуального состояния

Вариант №2. С подзапросом (PostgreSQL):



Кейс 1. Получение актуального состояния

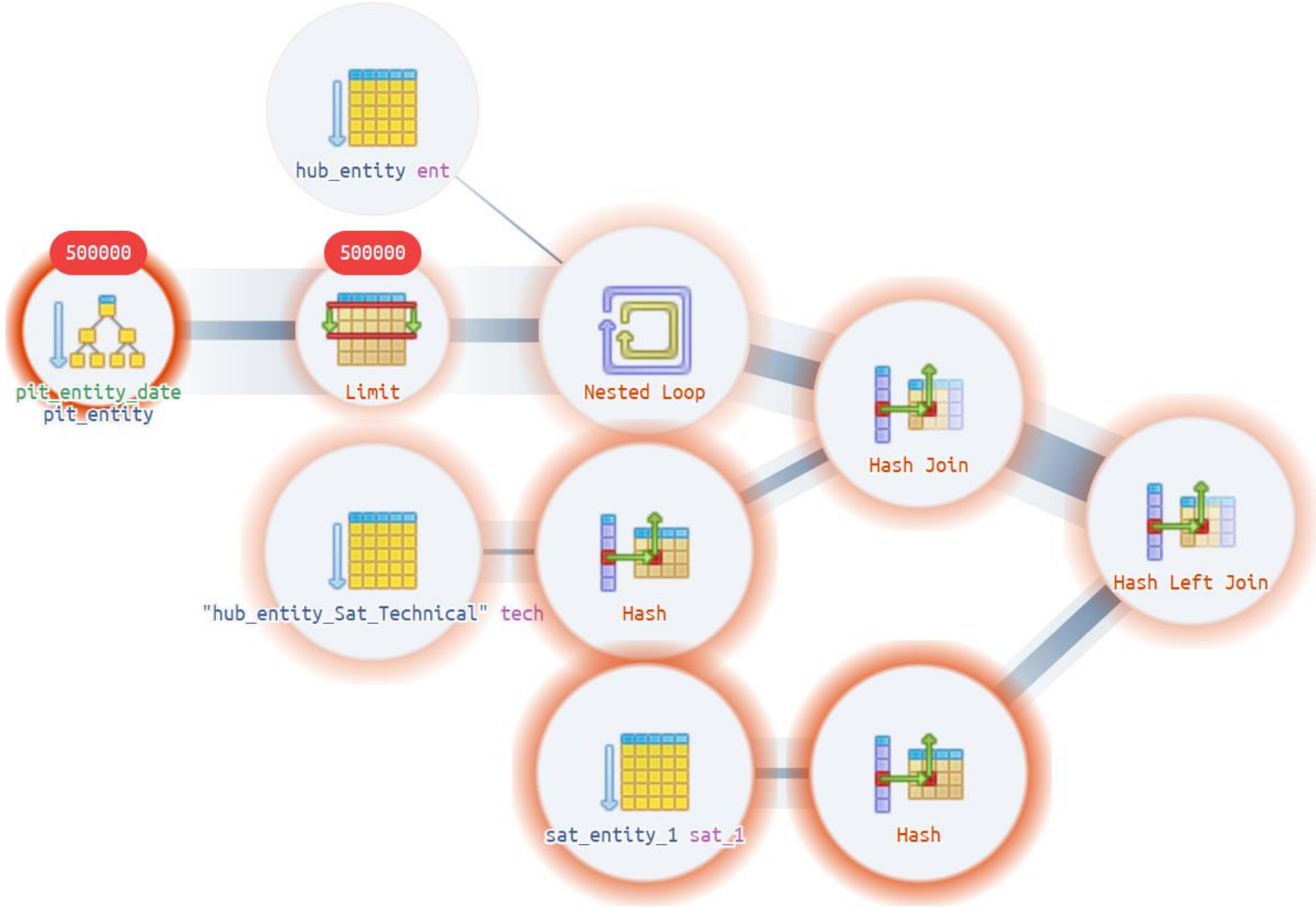
Вариант №3. С Point-In-Time таблицей (PostgreSQL):

```
select
   ent.id, ent.bk, sat_1.f1
from
   vault.hub_entity ent

   left join lateral
   (
     select
       *
     from
       vault.pit_entity
     where
       entity_id = ent.id
     order by
       load_date desc
     limit 1
   ) pit on true

   inner join vault."hub_entity_Sat_Technical" tech on
     tech.id = pit.sat_t_id

   left join vault.sat_entity_1 sat_1 on
     sat_1.id = pit.sat_1_id
;
```

Последние записи PIT

Кейс 1. Получение актуального состояния

Вариант №3. С Point-In-Time таблицей (PostgreSQL):



Кейс 1. Получение актуального состояния

Способ	Сложность генерируемой логики SQL (0-5)	Поддержка insert-only	Сложность поддержки доп. объектов	Время выполнения SQL	Режим выполнения	Нагрузка на CPU по длительности	Нагрузка на CPU по величине
1. Доп. атрибут, обновляемый при накате изменений	1	нет	2	1,2 сек	параллельный	короткая	высокая
2. Подзапрос с логикой определения последней записи для каждого идентификатора сущности	4	да	0	5,4 сек	последовательный	продолжительная	низкая
3. Использовать PIT-таблицу	2	Да	5	8,2 сек	последовательный	средняя	низкая

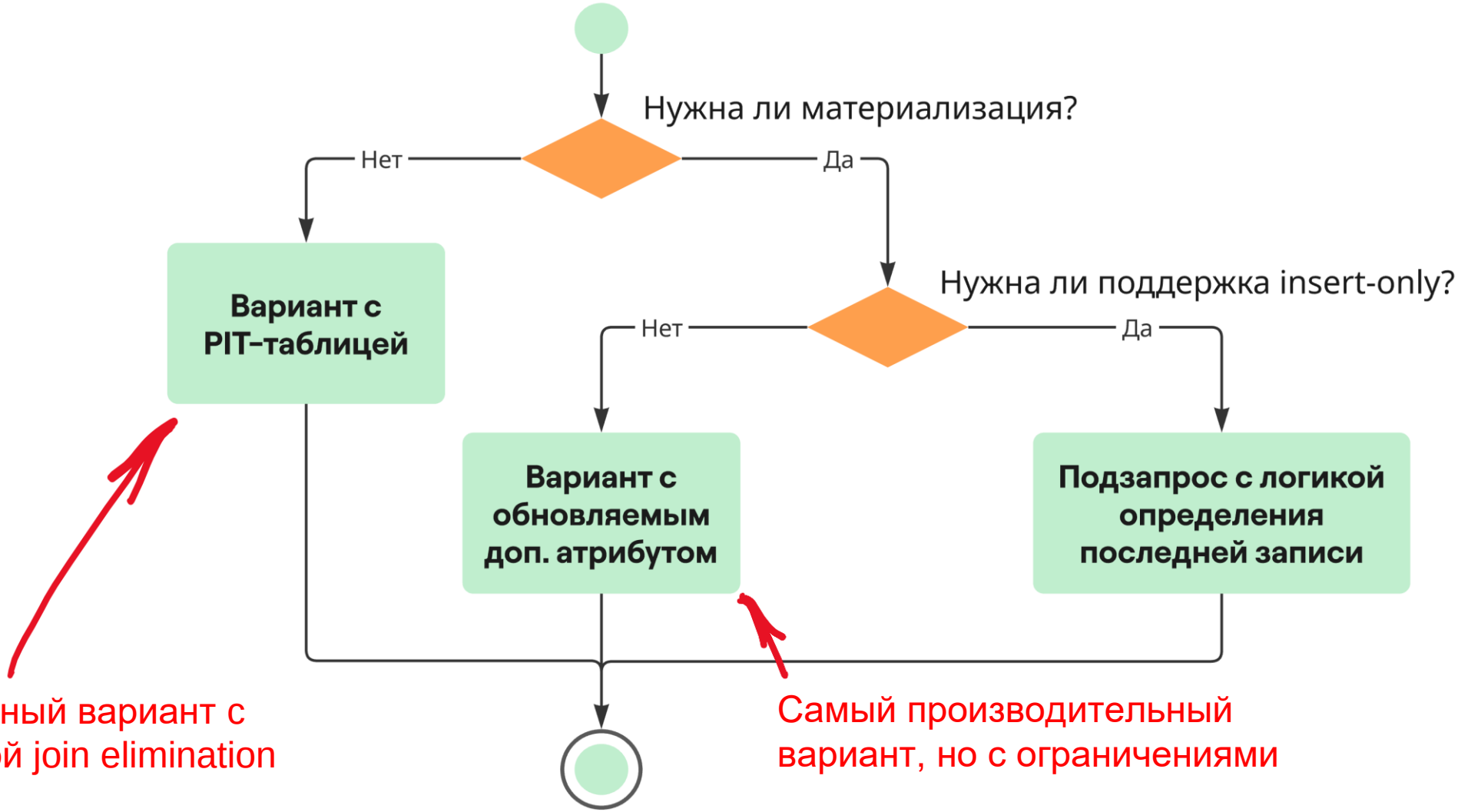
Выводы

- В случае материализации использование PIT для этих запросов не оправдано по соображениям производительности
- Если insert-only не критичен (PostgreSQL, Greenplum без append only) – имеет смысл смотреть в сторону использования флага индикации последней записи, но с ограничением по количеству потоков на запрос в случае большого количества запросов в параллели
- При использовании business vault без материализации выбираем вариант с PIT-таблицей, т.к. работает Join Elimination

Тестовый стенд: 4 ядра, 4 ГБ

Кейс 1. Получение актуального состояния

Алгоритм выбора варианта реализации при использовании представлений или табличных функций в Business Vault



Единственный вариант с поддержкой join elimination

Самый производительный вариант, но с ограничениями

Перенос логики сборки объектов Business Vault с PostgreSQL на Greenplum

Если сателлитов/линков >15, меняем настройки:

```
set optimizer = off;
set join_collapse_limit = 15;
set from_collapse_limit = 15;
```

Перенос логики сборки объектов Business Vault с PostgreSQL на Greenplum

PostgreSQL

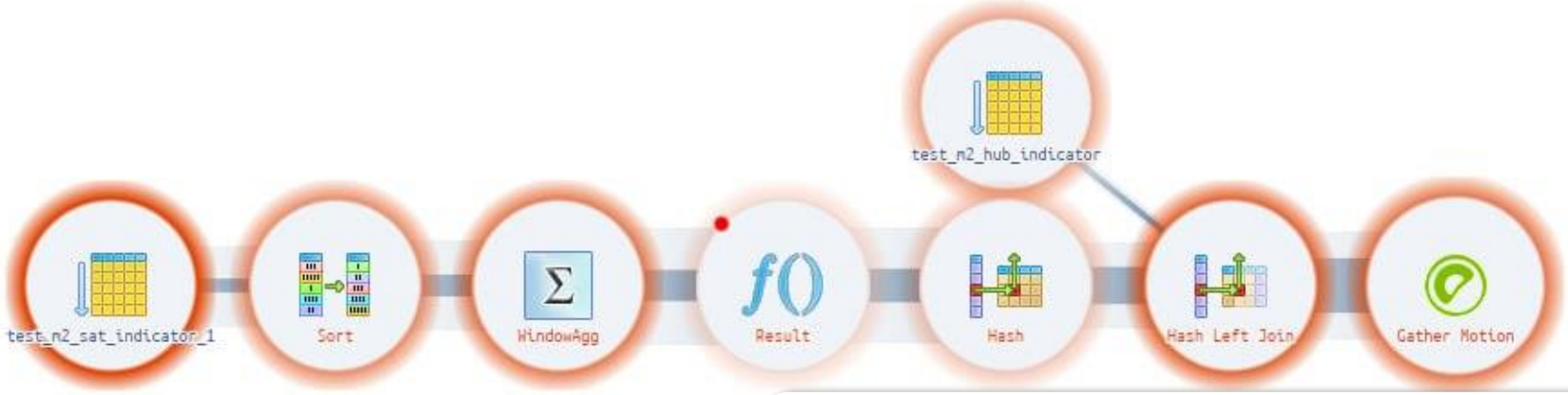
```
left join lateral
(
   select sat.*
   from
     mmd.sat_indicator sat
   where sat.entity_id = ent.id
   order by sat.load_date desc
   limit 1
) sat on true
```

Greenplum

```
left join
(
   select * from
   (
      select sat.*,
        row_number() over
          (partition by sat.entity_id
           order by sat.load_date desc) as is_actual
      from mmd.sat_indicator sat
   ) sat
   where sat.is_actual = 1
) sat on sat.entity_id = ent.id
```

lateral join -> row_number() – в 311 раз быстрее (на конкретном кейсе)

Перенос логики сборки объектов Business Vault с PostgreSQL на Greenplum



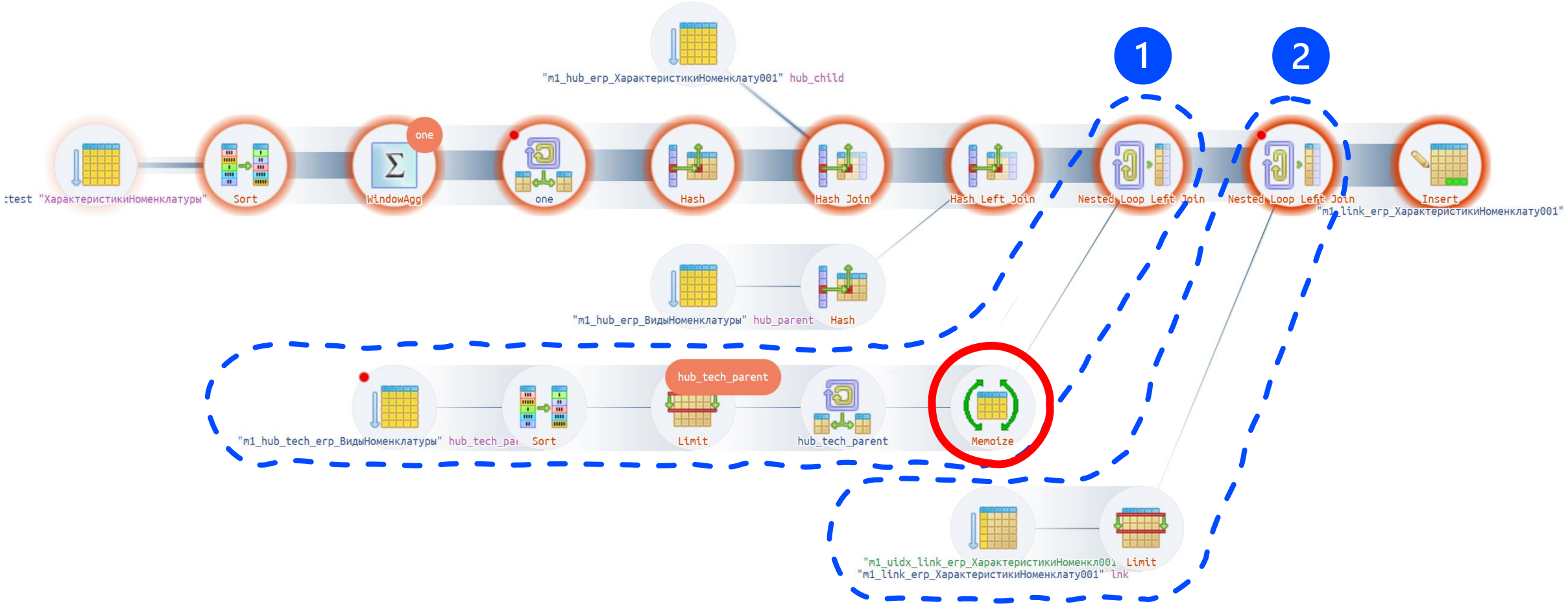
5 ms vs 1867 ms

node, ms	tree, ms	rows	
	3.827	3'979	итоговые результаты (78KB = rows=3'979 x width=20)
.568	3.827	3'979	Gather Motion
.656	3.259	1'005	-> Hash Left Join
.372		1'005	-> Seq Scan on test_m2_hub_indicator
.224	2.231	1'005 ▲	-> Hash
.148	2.007	1'005 ▲	-> Result
.552	1.859	1'014	-> WindowAgg
.477	1.307	1'014	-> Sort
.830		1'014	-> Seq Scan on test_m2_sat_indicator_1
4.438			EP Planning time
			Memory used
			Optimizer
1.388	5.215		Execution time

Замедление ETL при сборке Business Vault в PostgreSQL

```
left join lateral
(
    select
        *
    from
        "vault"."m1_hub_tech_erp_ВидыНоменклатуры" hub_tech_parent
    where
        hub_tech_parent.entity_id = "hub_parent"."id"
    order by
        load_date_norm desc
    limit 1
) hub_tech_parent on true
```

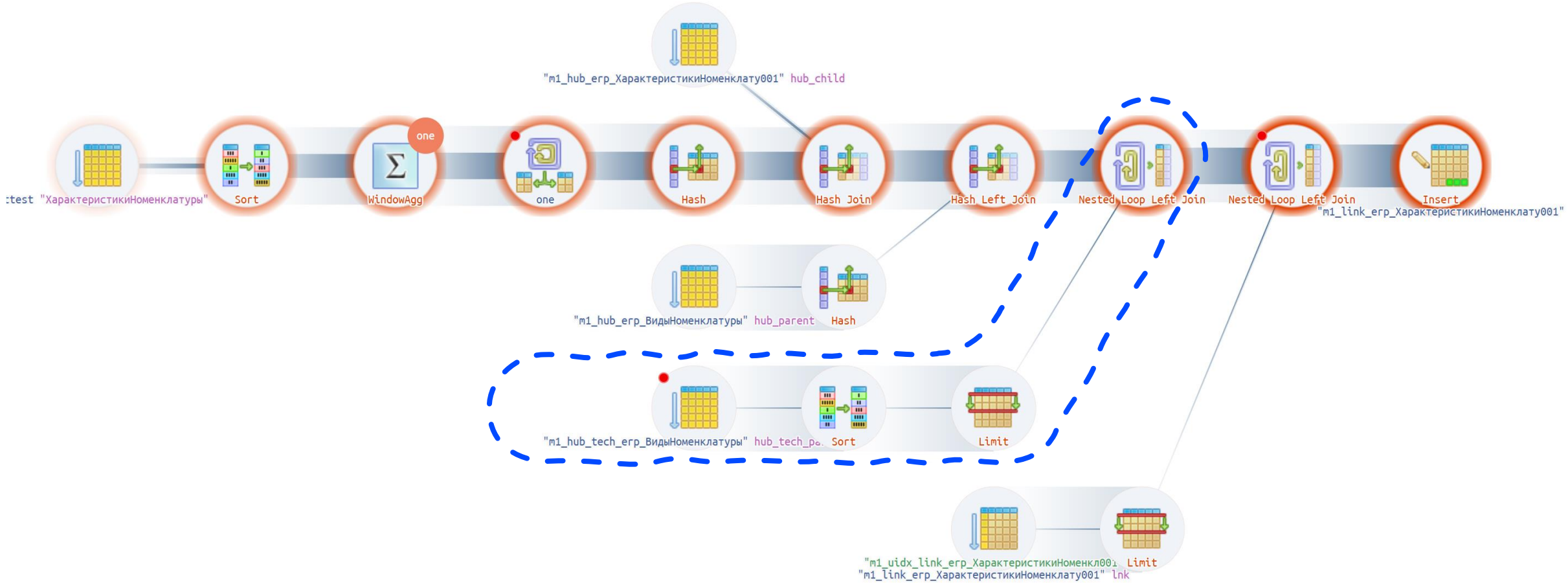
Замедление ETL при сборке Business Vault в PostgreSQL



1
```
create unique index "m1_uidx_hub_tech_erp_ВидыНоменклатуры"
  on vault."m1_hub_tech_erp_ВидыНоменклатуры" (entity_id, load_date_norm) include (is_exists, id);
```

2
```
create unique index "m1_uidx_link_erp_ХарактеристикиНоменкл001"
  on vault."m1_link_erp_ХарактеристикиНоменклату001" (child_id, load_date_norm);
```

Замедление ETL при сборке Business Vault в PostgreSQL




```
set enable_memoize = off;
```

Кейс 2. История изменений в формате SCD2

Задача

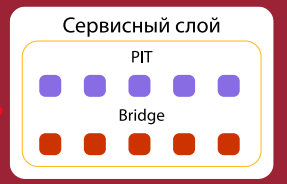
- Получить историю изменений в формате SCD2

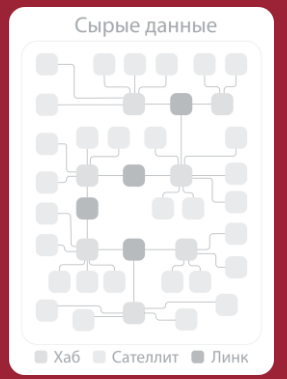
Применение

- В процедурах материализации истории изменений
- В представлениях истории изменений слоя business-vault
- Похожий код в процедурах актуализации PIT

Варианты реализации

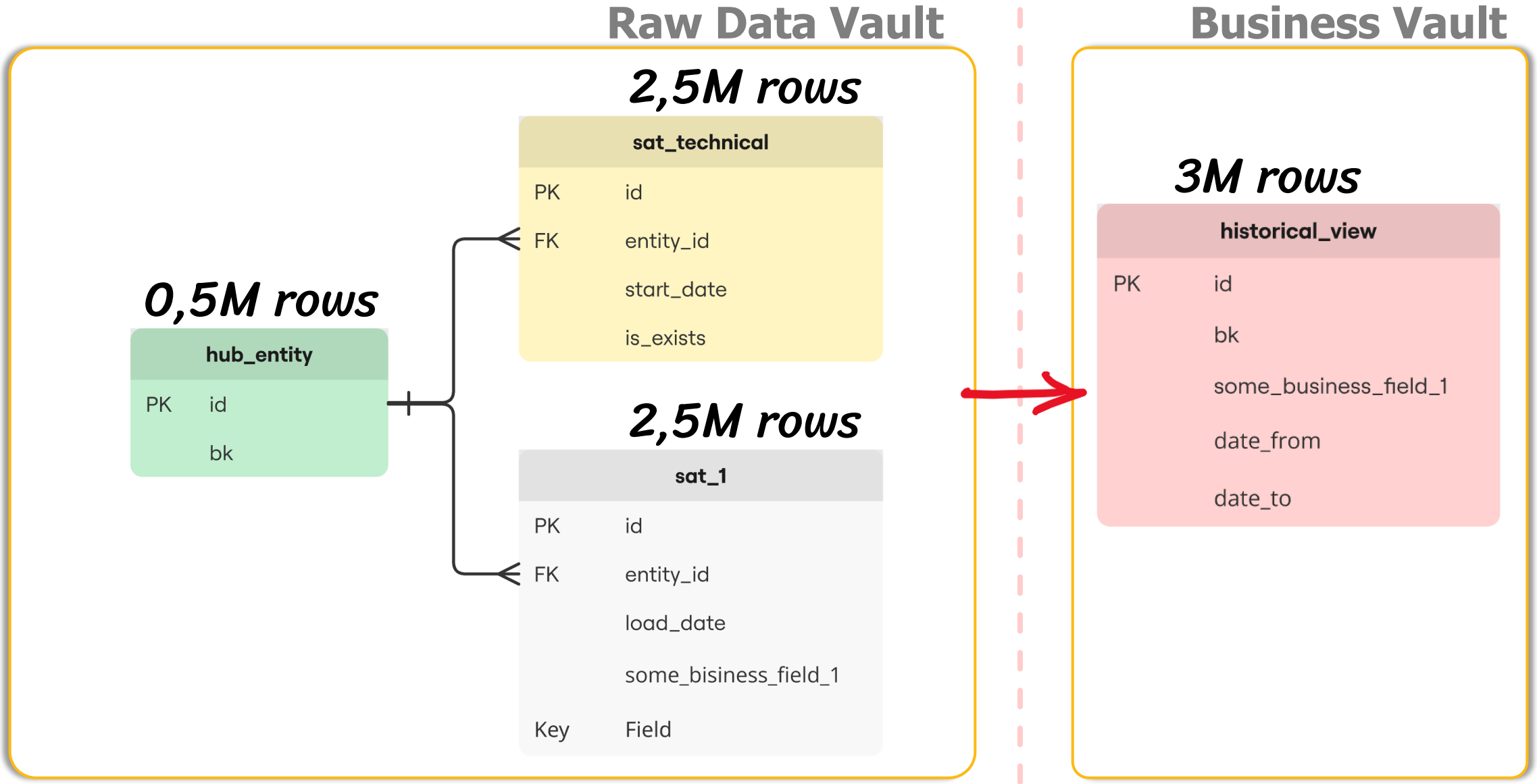
1. Использовать PIT-таблицу
2. Логика с подзапросами и определением т.н. **Unified Timeline**





Кейс 2. История изменений в формате SCD2

Модельная задача



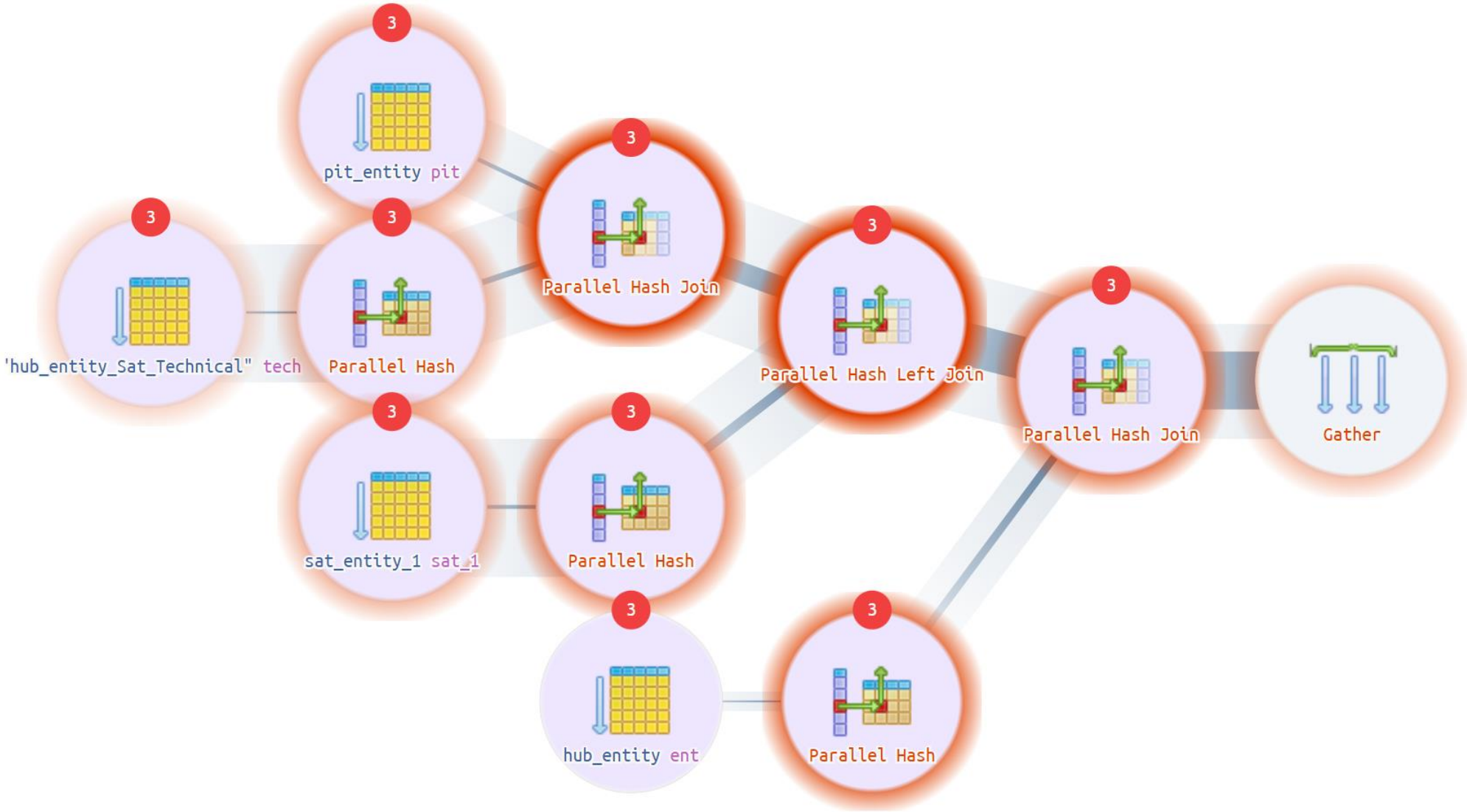
Кейс 2. История изменений в формате SCD2

Вариант с PIT

```
select
   ent.id, ent.bk, sat_1.f1
from
   vault.hub_entity ent

   left join vault.pit_entity pit on
     pit.entity_id = ent.id

   inner join vault."hub_entity_Sat_Technical" tech on
     tech.id = pit.sat_t_id

   left join vault.sat_entity_1 sat_1 on
     sat_1.id = pit.sat_1_id
;
```

Кейс 2. История изменений в формате SCD2

Вариант с PIT



Кейс 2. История изменений в формате SCD2

Вариант с Unified Timeline

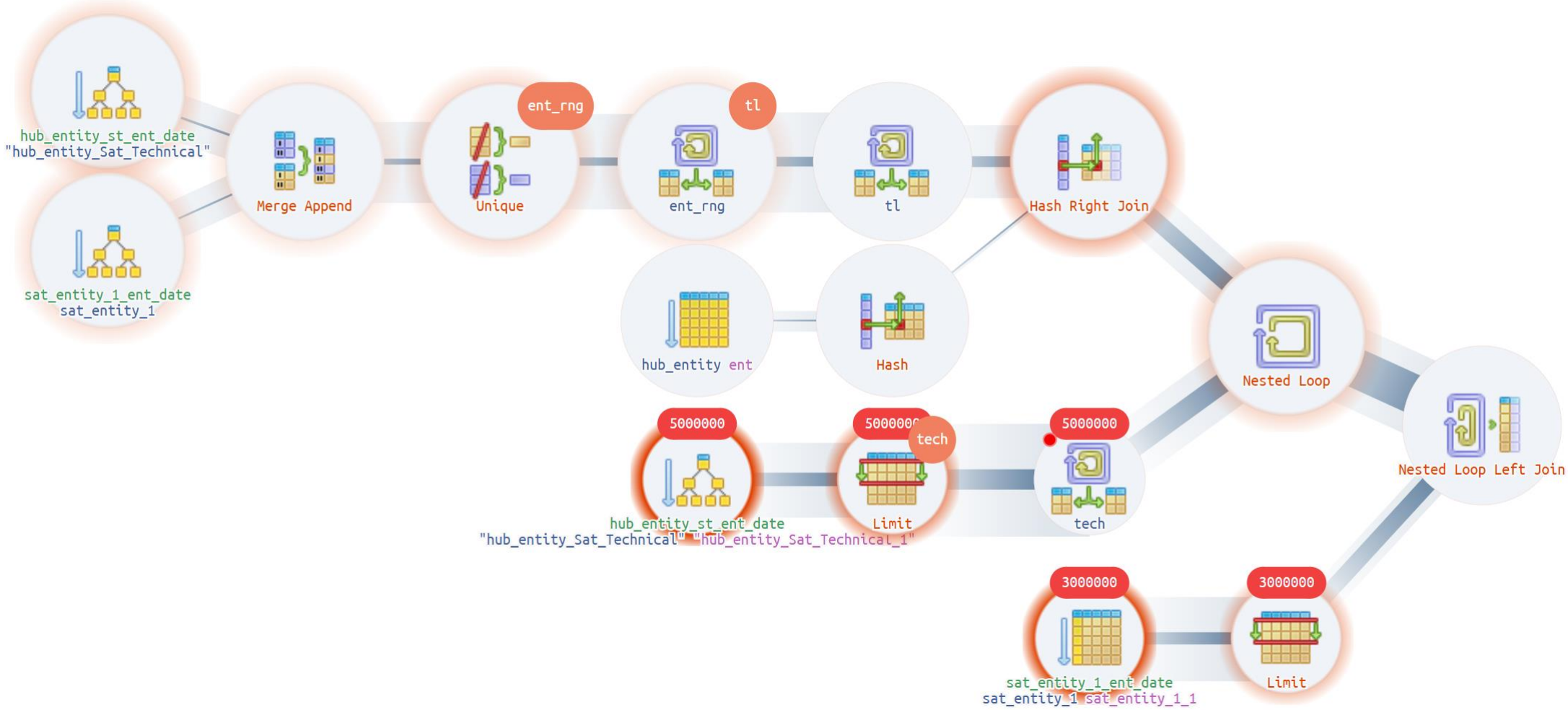
```
with unified_timeline as
(
   select
      entity_id
      ,date_from
      ,lead(date_from, 1, '99990101'::timestamp)
       over (partition by entity_id order by
date_from) as date_to
   from
   (
      select
         entity_id, start_date as date_from
      from
         vault."hub_entity_Sat_Technical"
      union
      select
         entity_id, load_date
      from
         vault.sat_entity_1
   ) ent_rng
)
```

```
select
   ent.id as entity_id
   ,ent.bk
   ,tl.date_from
   ,sat_1.f1
from
   vault.hub_entity ent
   left join unified_timeline tl on
      tl.entity_id = ent.id
   inner join lateral
   (
      select
         is_exists
      from
         vault."hub_entity_Sat_Technical"
      where
         entity_id = ent.id
         and
         start_date <= tl.date_from
      order by
         start_date desc
      limit 1
   ) tech on is_exists
```

```
left join lateral
   (
      select
         *
      from
         vault.sat_entity_1
      where
         entity_id = ent.id
         and
         load_date <= tl.date_from
      order by
         load_date desc
      limit 1
   ) sat_1 on true
```

Кейс 2. История изменений в формате SCD2

Вариант с Unified Timeline



Кейс 2. История изменений в формате SCD2

Вариант с Unified Timeline

IC #7 IC #8 #13 IOSB | tilemap | piechart

#	node, ms	tree, ms	rows	loops	
		41'425.520	3'000'000		итоговые результаты (140MB = rows=3'000'000 x width=49)
0	173.174	41'425.520	3'000'000		Nested Loop Left Join
1	1'981.440	29'252.346	3'000'000		-> Nested Loop
2	3'641.506	12'270.906	5'000'000		-> Hash Right Join
3	876.242	8'028.057	5'000'000		-> Subquery Scan on tl
4	1'092.903	7'151.815	5'000'000		-> Subquery Scan on ent_rng
5	1'704.865	6'058.912	5'000'000		-> Unique
6	1'098.923	4'354.047	5'000'000		-> Merge Append
7	1'807.186		2'500'000	IC	-> Index Only Scan using hub_entity_st_ent_date on "hub_entity_Sat_Technical"
8	1'447.938		2'500'000	IC	-> Index Only Scan using sat_entity_1_ent_date on sat_entity_1
9	488.538	601.343	500'000		-> Hash
10	112.805		500'000		-> Seq Scan on hub_entity ent
11		15'000.000	5'000'000	5'000'000	-> Subquery Scan on tech
12	5'000.000	15'000.000	5'000'000	5'000'000	-> Limit
13	10'000.000		5'000'000	5'000'000	-> Index Only Scan Backward using hub_entity_st_ent_date on "hub_entity_Sat_Technical" "hub_entity_Sat_Technical_1"
14	3'000.000	12'000.000	3'000'000	3'000'000	-> Limit
15	9'000.000		3'000'000	3'000'000	-> Index Scan Backward using sat_entity_1_ent_date on sat_entity_1 sat_entity_1_1
	.844				Planning Time
	297.540				JIT
	224.824	41'650.344			Execution Time: 41'650.344 ms

Кейс 2. История изменений в формате SCD2

Способ	Сложность генерируемой логики SQL (0-5)	Поддержка insert-only	Время выполнения SQL	Режим выполнения	Нагрузка на CPU по длительности	Нагрузка на CPU по величине
Использовать PIT-таблицу	2	да	4,2 сек	параллельный	короткая	высокая
Сложная логика с подзапросами и определением т.н. **Unified Timeline**	4	да	41,6 сек	последовательный	продолжительная	низкая

Выводы

- Для историчных представлений применение PIT-таблицы предпочтительно
- При включенной материализации применение PIT-таблиц требует инкрементального обновления, в противном случае её применение не целесообразно по причине двойной материализации
- При использовании business vault без материализации вариант с PIT-таблицей ваш выбор, т.к. работает Join Elimination

Тестовый стенд: 4 ядра, 4 ГБ

Кейс 3. Медленное соединение таблиц

Замедление построения Data Lineage в PostgreSQL и Greenplum

Если PostgreSQL/Greenplum не может выполнить соединение таблиц с помощью hash join, то обычно в плане запроса появляется loop join с перебором “каждый с каждым”. Даже на небольших объёмах это может приводить к существенной деградации производительности

Пример запроса с выражением для соединения таблиц, которое не может быть трансформировано в hash join:

```
select distinct on(src.table_name)
    inf.schemaname  as ods_table_schema
    ,inf.tablename  as ods_table_name
    ,src.*
from
    (
        select distinct on(table_name)
            server_instance
            ,db_name
            ,db_type
            ,schema_name
            ,table_name
            ,table_type
        from
            metadata.source_metadata
        order by
            table_name
    ) as src

    inner join pg_catalog.pg_tables inf on
        -- медленный вариант предиката, приводит к loop join
        inf.tablename ilike '%$_' || src.table_name escape '$'

where
    inf.schemaname like 'ods%'

order by
    src.table_name
    ,length(inf.tablename)
    ,inf.tablename
    ,inf.schemaname
```

Оптимизированный вариант запроса. После анализа данных выяснилось, что результат в данном конкретном кейсе эквивалентный, что может быть неверно в общем случае:

```
select distinct on(src.table_name)
    inf.schemaname  as ods_table_schema
    ,inf.tablename  as ods_table_name
    ,src.*
from
    (
        select distinct on(table_name)
            server_instance
            ,db_name
            ,db_type
            ,schema_name
            ,table_name
            ,table_type
        from
            metadata.source_metadata
        order by
            table_name
    ) as src

    inner join pg_catalog.pg_tables inf on
        -- приводит к loop join
        --inf.tablename ilike '%$_' || src.table_name escape '$'
        split_part(inf.tablename, '__', 2) = lower(src.table_name)

where
    inf.schemaname like 'ods%'

order by
    src.table_name
    ,length(inf.tablename)
    ,inf.tablename
    ,inf.schemaname
```

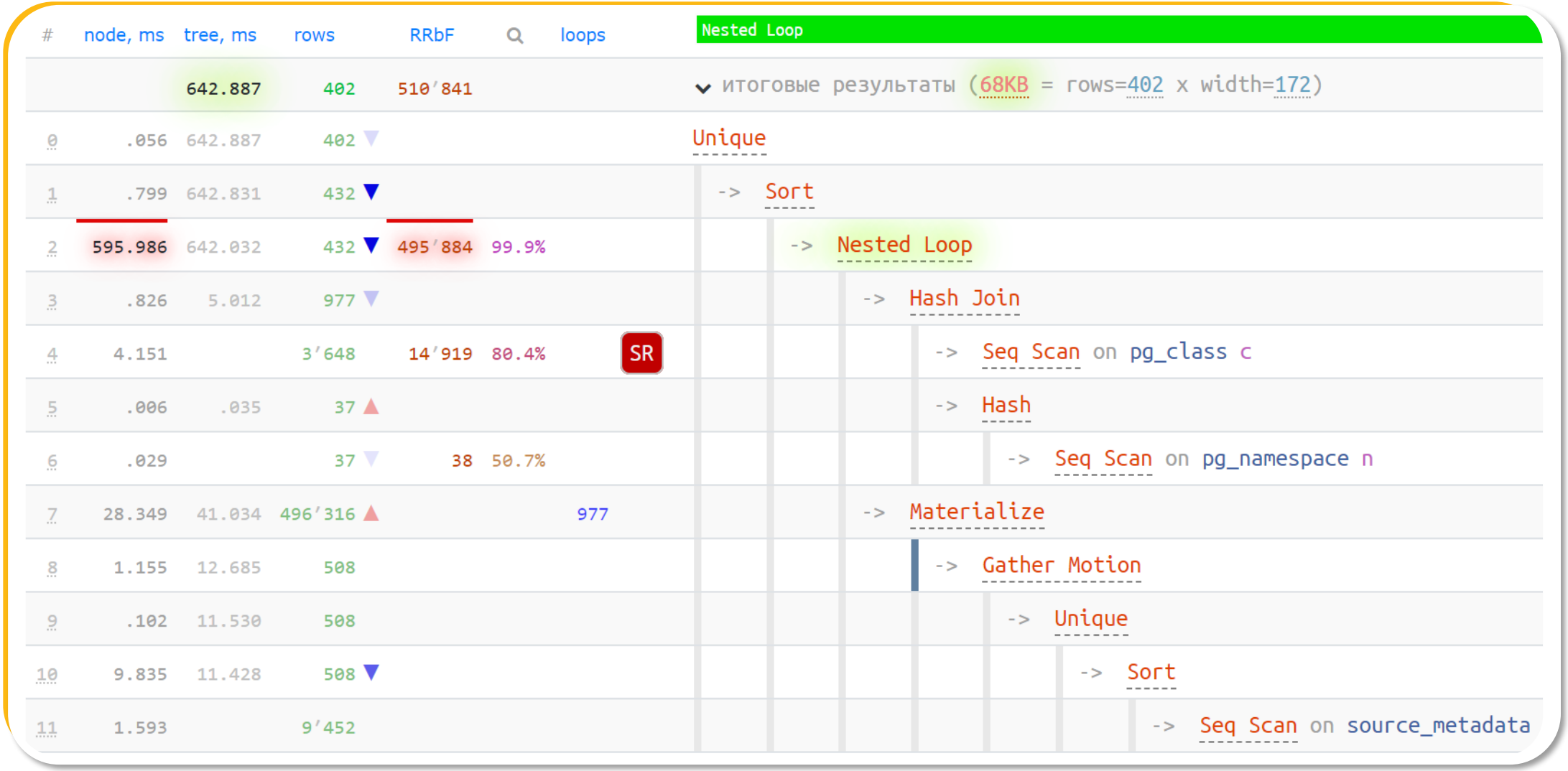
Кейс 3. Медленное соединение таблиц

```
22      inner join pg_catalog.pg_tables inf on
23          -- медленный вариант предиката, приводит к loop join
24          inf.tablename ilike '%$_' || src.table_name escape '$'
```

```
57      inner join pg_catalog.pg_tables inf on
58          -- приводит к loop join
59          --inf.tablename ilike '%$_' || src.table_name escape '$'
60          split_part(inf.tablename, '__', 2) = lower(src.table_name)
```

Кейс 3. Медленное соединение таблиц

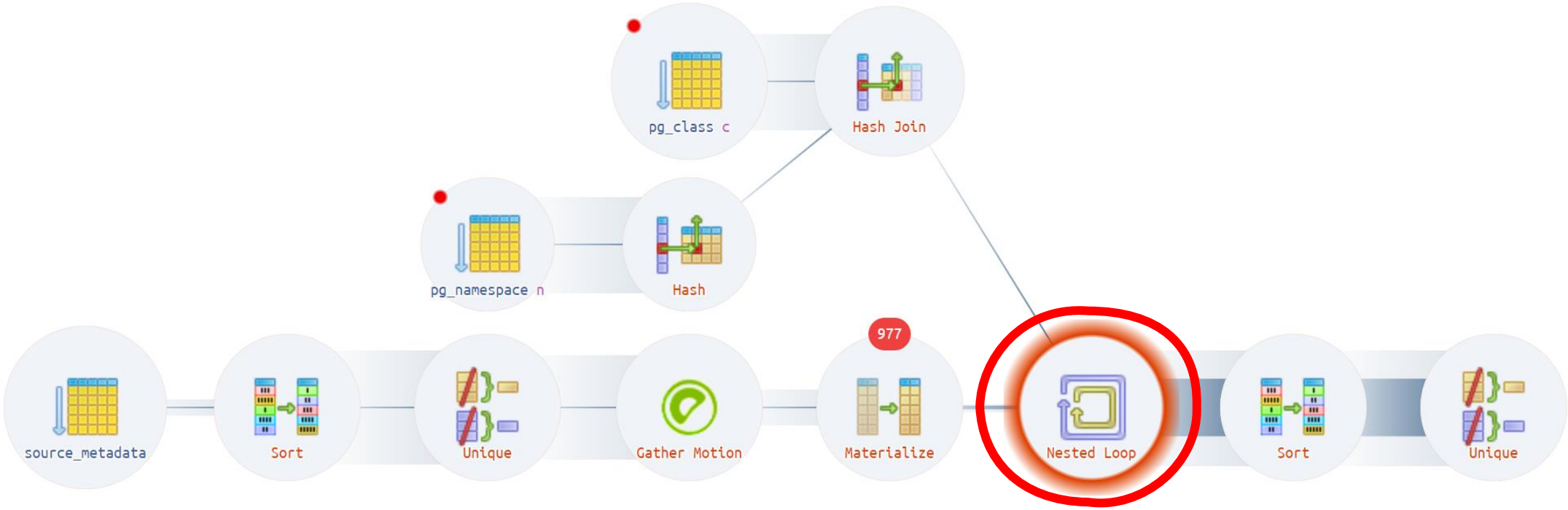
План оригинального запроса (см. план):



#	node, ms	tree, ms	rows	RRbF		loops	
		642.887	402	510'841			Nested Loop
							итоговые результаты (68KB = rows=402 x width=172)
0	.056	642.887	402				Unique
1	.799	642.831	432				-> Sort
2	595.986	642.032	432	495'884	99.9%		-> Nested Loop
3	.826	5.012	977				-> Hash Join
4	4.151		3'648	14'919	80.4%	SR	-> Seq Scan on pg_class c
5	.006	.035	37				-> Hash
6	.029		37	38	50.7%		-> Seq Scan on pg_namespace n
7	28.349	41.034	496'316			977	-> Materialize
8	1.155	12.685	508				-> Gather Motion
9	.102	11.530	508				-> Unique
10	9.835	11.428	508				-> Sort
11	1.593		9'452				-> Seq Scan on source_metadata

Кейс 3. Медленное соединение таблиц

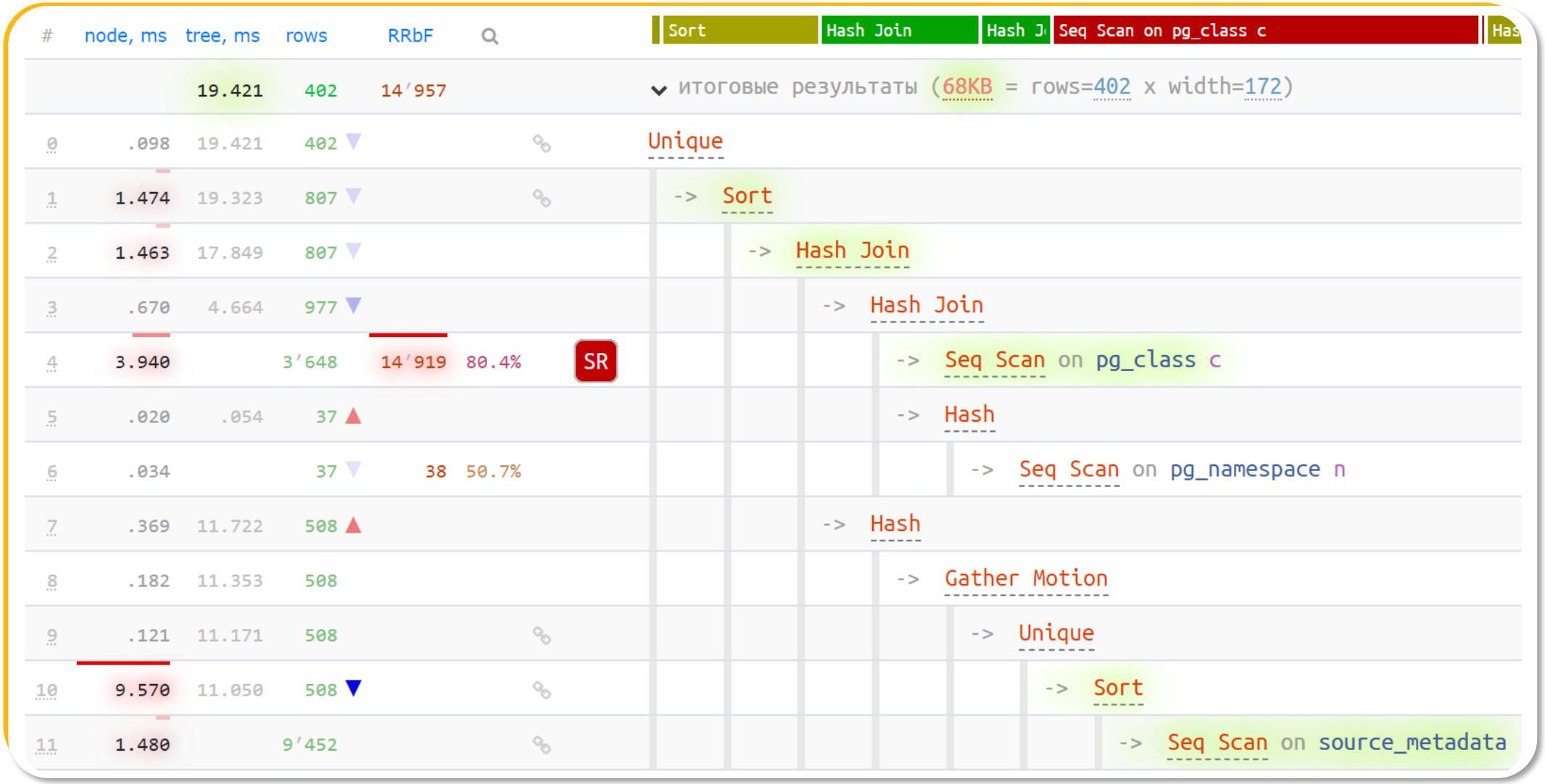
План оригинального запроса (см. план):



Узел Nested Loop выполняется 596 мсек, что составляет **более 90%** от общего времени выполнения запроса. При этом с помощью Nested Loop соединяются два сравнительно небольших набора записей: 508 и 977 записей.

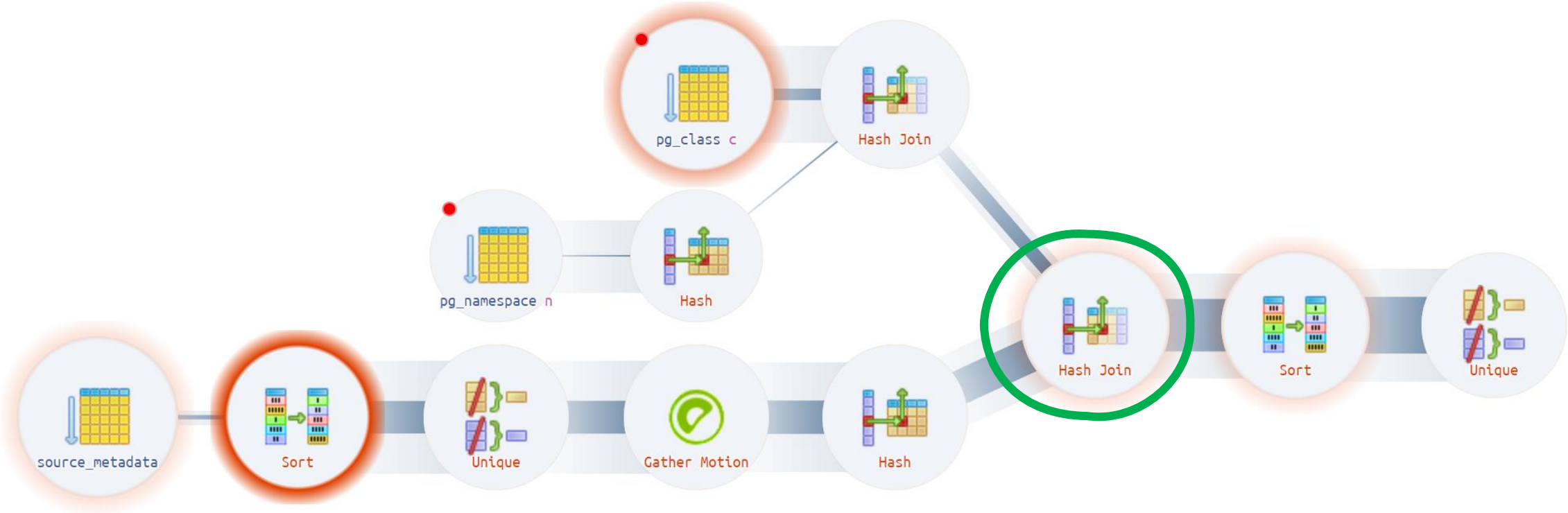
Кейс 3. Медленное соединение таблиц

План оптимизированного варианта (см. план) с предикатом под hash join:



Кейс 3. Медленное соединение таблиц

План оптимизированного варианта (см. план) с предикатом под hash join:



Оптимизированный запрос быстрее оригинального **в 32 раза** – 640 мсек оригинальный запрос и 20 мсек оптимизированный

Кейс 4. Перенос бизнес-логики с SMP на MPP

Медленная работа при запросе сателлита в Greenplum

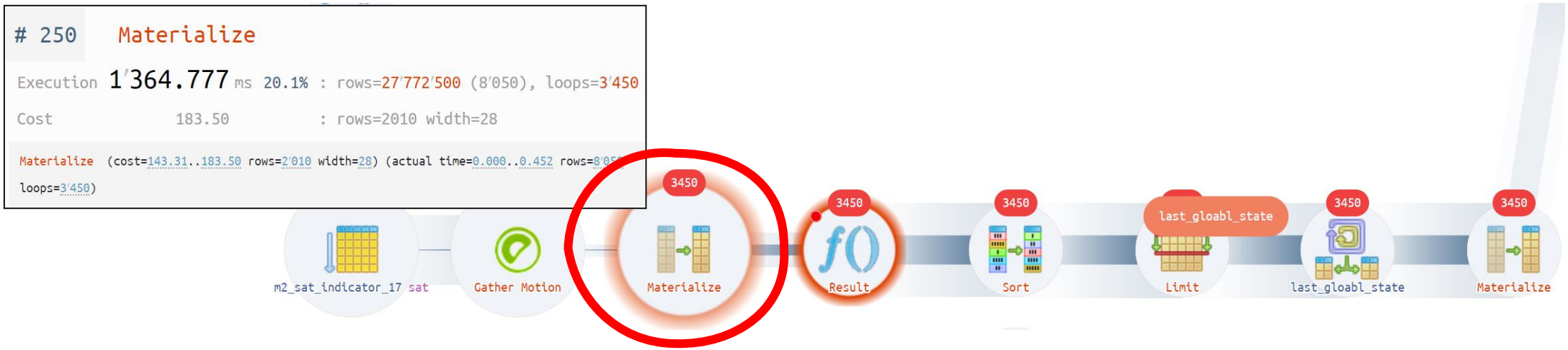
! В отличие от PostgreSQL подзапрос `last_global_state` на Greenplum выполняется на порядки медленнее

```
383     -- системное поле (логин пользователя, измненившего запись)
384     left join lateral
385     (
386         select
387             data_modified_by as modified_by
388         from
389             mmd_v.m2_sat_indicator_17 sat
390         where
391             data_modified_by not in
392             (
393                 'bondarev_bi'
394                 ,'proschenkov_aa'
395             )
396             and
397             sat.entity_id = ind.id
398             and
399             sat.load_date_norm <= global_state.load_date_norm
400         order by
401             load_date_norm desc
402         limit 1
403     ) last_gloabl_state on true
```

Кейс 4. Перенос бизнес-логики с SMP на MPP

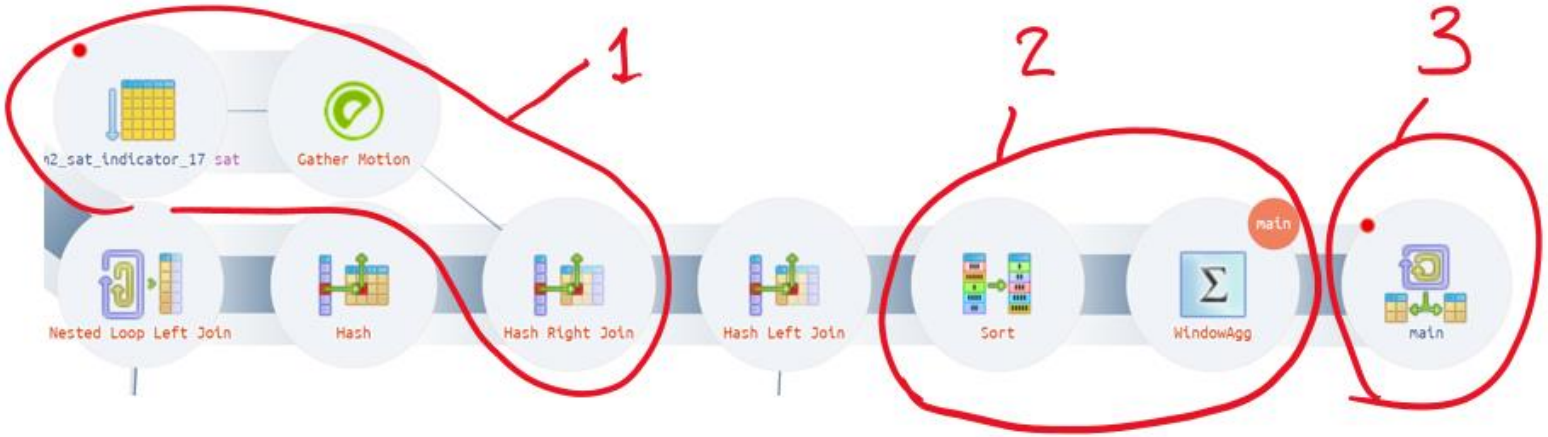
может приводить к плохому плану выполнения (см. план):

```
245    3.450  4'191.750      3'450   3'450   ->  Materialize
246           4'188.300      3'450   3'450       ->  Subquery Scan on last_gloabl_state
247    3.450  4'188.300      3'450   3'450           ->  Limit
248   13.800  4'184.850      3'450 ▼ 3'450               ->  Sort
249  2'611.650 4'171.050     3'450 ▼ 3'450                   ->  Result
250  1'364.777 1'559.400 27'772'500 ▲ 3'450                       ->  Materialize  (cost=143.31..183.50 rows=2'010 width=28) (actual time=0.000..0.452 rows=8'050 loops=3'450)
251     52.778   194.623      8'050                                   ->  Gather Motion
252    141.845                8'050                                       ->  Seq Scan on m2_sat_indicator_17 sat
```



Кейс 4. Перенос бизнес-логики с SMP на MPP

В результате план запроса принимает следующий вид (см. план):

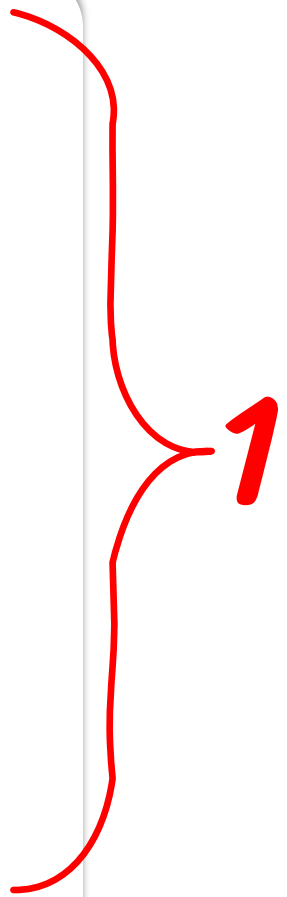



```sql
select
    *
from
(
    select
        ind.*
        --...
        ,max
        (
            case
                when last_gloabal_state.load_date_norm <= global_state.load_date_norm
                then last_gloabal_state.load_date_norm
            end
        ) over (partition by ind.id)       as max_last_gloabal_load_date_norm
        ,last_gloabal_state.load_date_norm as last_gloabal_load_date_norm
    from
        mmd_v.m2_act_indicator ind

        --...

        left join lateral
        (
            select
                t.state
                ,t.code
                ,t.load_date
                ,t.load_date_norm
            from
            (
                --...
            ) t

            order by
                t.load_date desc nulls last
                ,t.priority desc nulls last
            limit 1

        ) global_state on true

        -- системное поле (логин пользователя, изменившего запись)
        left join
        (
            select
                entity_id
                ,load_date_norm
                ,data_modified_by as modified_by
            from
                mmd_v.m2_sat_indicator_17 sat
            where
                data_modified_by not in
                (
                    'bondarev_bi'
                    ,'proschenkov_aa'
                )
        ) last_gloabal_state on
            last_gloabal_state.entity_id = ind.id
) main

where
    max_last_gloabal_load_date_norm is null
    or
    max_last_gloabal_load_date_norm = last_gloabal_load_date_norm
```

Кейс 4. Перенос бизнес-логики с SMP на MPP

В результате план запроса принимает следующий вид (см. план):

```
41          left join
42          (
43              select
44                  entity_id
45                  ,load_date_norm
46                  ,data_modified_by as modified_by
47              from
48                  mmd_v.m2_sat_indicator_17 sat
49              where
50                  data_modified_by not in
51                  (
52                      'bondarev_bi'
53                      ,'proschenkov_aa'
54                  )
55          ) last_gloabal_state on
56              last_gloabal_state.entity_id = ind.id
57  ) main
```



Кейс 4. Перенос бизнес-логики с SMP на MPP

В результате план запроса принимает следующий вид (см. план):

```
8           ,max
9           (
10              case
11                  when last_gloabal_state.load_date_norm <= global_state.load_date_norm
12                  then last_gloabal_state.load_date_norm
13              end
14          ) over (partition by ind.id)       as max_last_gloabal_load_date_norm
```

2

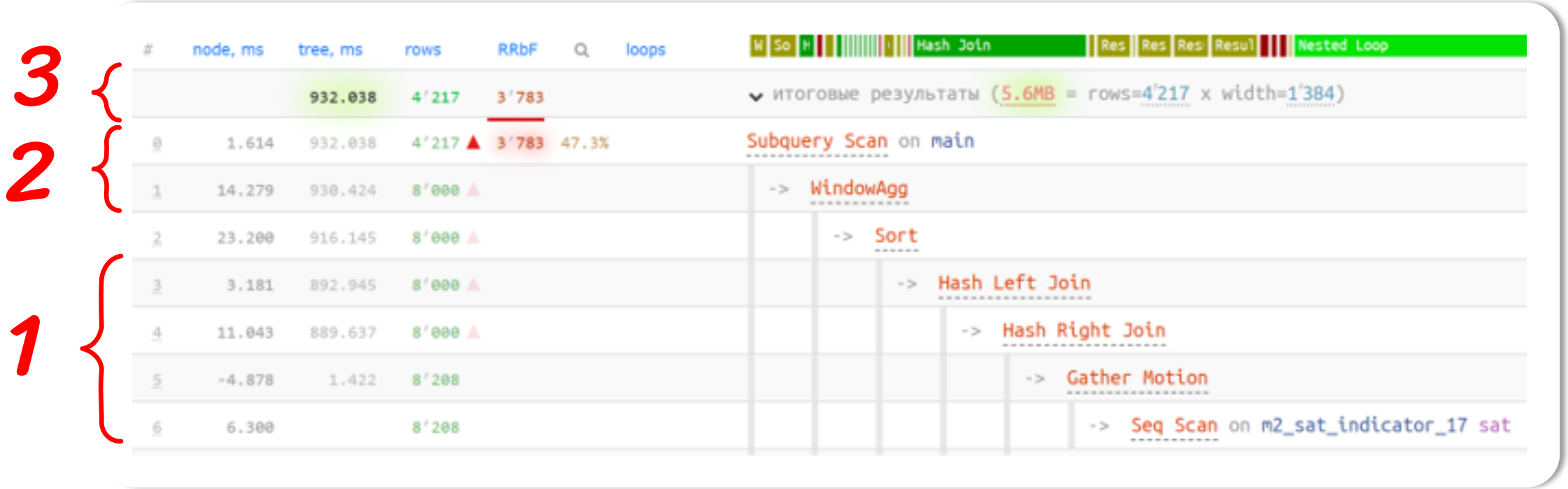
Кейс 4. Перенос бизнес-логики с SMP на MPP

В результате план запроса принимает следующий вид (см. план):

```
58
59  where
60      max_last_gloabal_load_date_norm is null
61      or
62      max_last_gloabal_load_date_norm = last_gloabal_load_date_norm
```

3

Кейс 4. Перенос бизнес-логики с SMP на MPP



Первоначальный подзапрос выполнялся ~4 секунды (4 191 мсек), эквивалентные ему функциональные блоки выполняются **в 80 раз быстрее** – ~0,05 сек (50 мсек)

Кейс 5. Greenplum “union all” vs “in”, “or”

Проблема: код **варианта с “in”** не может утилизировать индексы

Подход с “in”

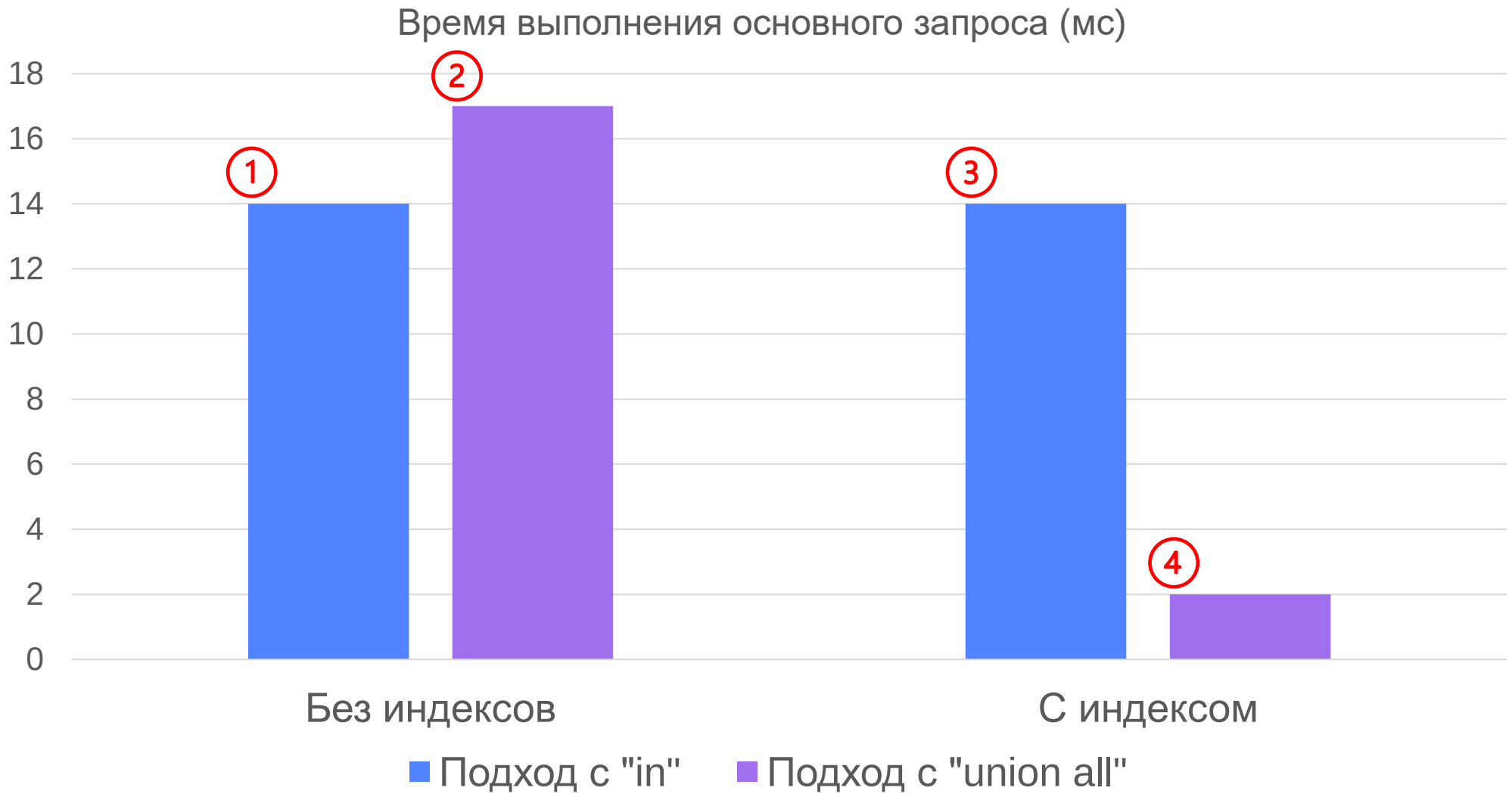
```
select
   v.dt
   ,case v.key_id
      when v_key_id_1 then v.value
      else -v.value
   end as value
from
   ods_mesdb_gra.gra_rep__key_value v
where
   v.key_id in
   (
      v_key_id_1
      ,v_key_id_2
   )
   and
   v.shift  = 0
```

Подход с “union all”

```
select
   v.dt , v.value
from
   ods_mesdb_gra.gra_rep__key_value v
where
   v.key_id = v_key_id_1
   and
   v.shift  = 0
union all
select
   v.dt, -v.value
from
   ods_mesdb_gra.gra_rep__key_value v
where
   v.key_id = v_key_id_2
   and
   v.shift  = 0
```

Кейс 5. Greenplum “union all” vs “in”, “or”

Сравнение производительности sql-кода различных подходов



Кейс 5. Greenplum “union all” vs “in”, “or”

Выводы

1. Без создания дополнительных индексов время выполнения sql-запроса с “union all” на 3 мсек (на 18%) больше, чем время выполнения соответствующего запроса с “in”.
2. Добавление индекса не меняет ситуацию с подходом “in”, при этом ускоряет sql-код с “union all” до 2-х мсек, делая вызов в 7 раз быстрее, чем пример с “in”.
3. Запросы с “in” (в отличии от “union all”) не поддаются оптимизации с помощью индексов.
4. Без использования индексов время выполнения запросов с “union all” не существенно хуже (на десятки процентов), чем аналогичных запросов с “in”.
5. При наличии правильных индексов время выполнения запросов с “union all” во много раз лучше, чем запросов с “in”.

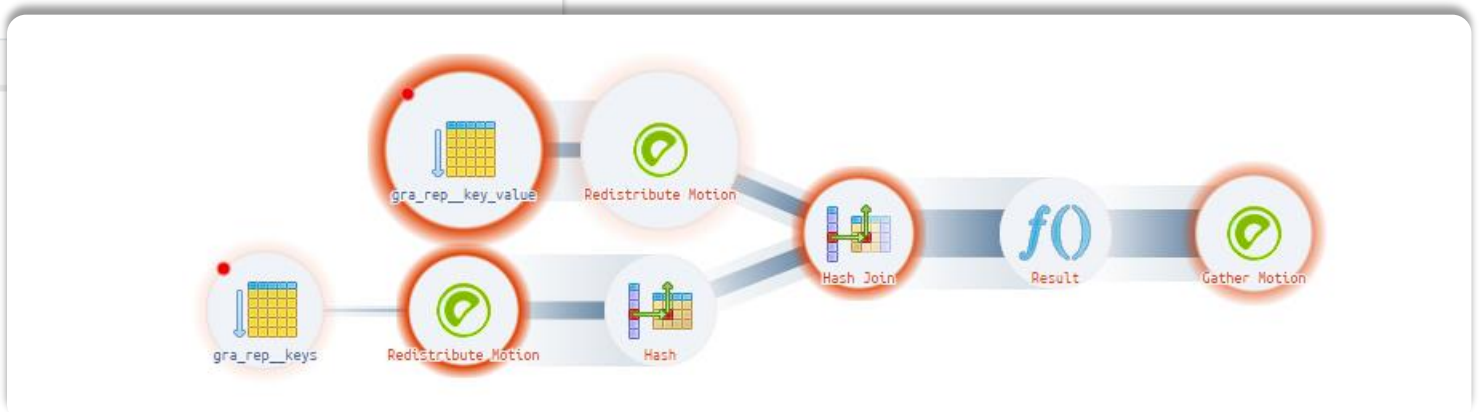
С точки зрения наглядности, простоты читаемости, удобства отладки, сопровождаемости и возможностей оптимизации **код с “union all” предпочтительнее, чем код с “in”**

Кейс 5. Greenplum “union all” vs “in”, “or”

План запроса с “in”. Вариант с явным обращением к таблице с ключами, наличие индексов не влияет на формируемый план запроса.

#	node, ms	tree, ms	rows	Gather Motion	Hash Join	Rec	Seq Scan on gra_rep__key_value	Redistribute Motion	Sec
		3.634	2	итоговые результаты (24B = rows=2 x width=12)					
0	.551	3.634	2 ▼	Gather Motion					
1	.021	3.083	1 ▼	-> Result					
2	.843	3.062	1 ▼	-> Hash Join					
3	.113	1.068	1’132 ▲	-> Redistribute Motion					
4	.955		1’032	-> Seq Scan on gra_rep__key_value					
5	.004	1.151	1 ▼	-> Hash					
6	1.029	1.147	1 ▼	-> Redistribute Motion					
7	.118		2 ▼	-> Seq Scan on gra_rep__keys					
	9.126		EP	Planning time					
				Memory used					
				Optimizer					
	10.367	14.001		Execution time					

① и ③

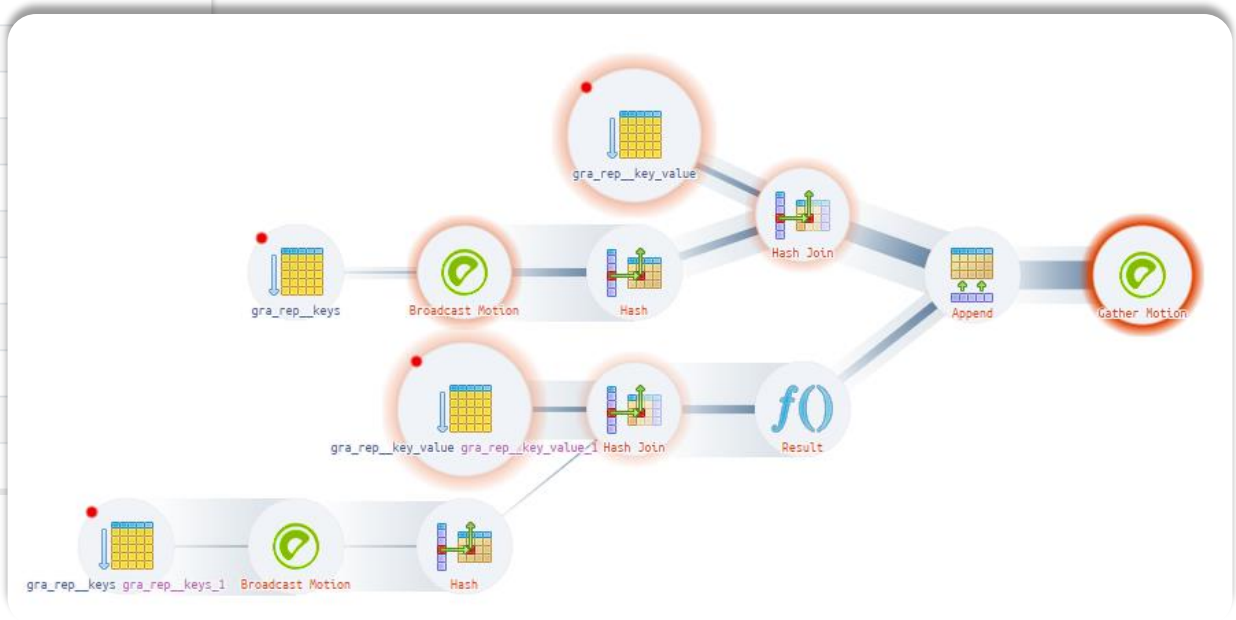


Кейс 5. Greenplum “union all” vs “in”, “or”

План запроса с “union all”. Вариант с явным обращением к таблице с ключами, без индексов.

#	node, ms	tree, ms	rows	Gather Motion	Hash Join	Seq Scan on gra	Broadcast Motion	Sc	Hash Jo	Seq Scan on	Sc
		6.995	2	итоговые результаты (16B = rows=2 x width=8)							
0	2.865	6.995	2	Gather Motion							
1	.037	4.130	1	-> Append							
2	.693	2.810	1	-> Hash Join							
3	.897		1’032	-> Seq Scan on gra_rep__key_value							
4	.003	1.220	1	-> Hash							
5	1.054	1.217	1	-> Broadcast Motion							
6	.163		1	-> Seq Scan on gra_rep__keys							
7	.037	1.283	1	-> Result							
8	.480	1.246	1	CC -> Hash Join							
9	.752		1’032	-> Seq Scan on gra_rep__key_value gra_rep__key_value_1							
10	.001	.014	1	-> Hash							
11	-0.140	.013	1	-> Broadcast Motion							
12	.153		1	-> Seq Scan on gra_rep__keys gra_rep__keys_1							
	16.026			EP Planning time							
				Memory used							
				Optimizer							
	10.309	17.304		Execution time							

②

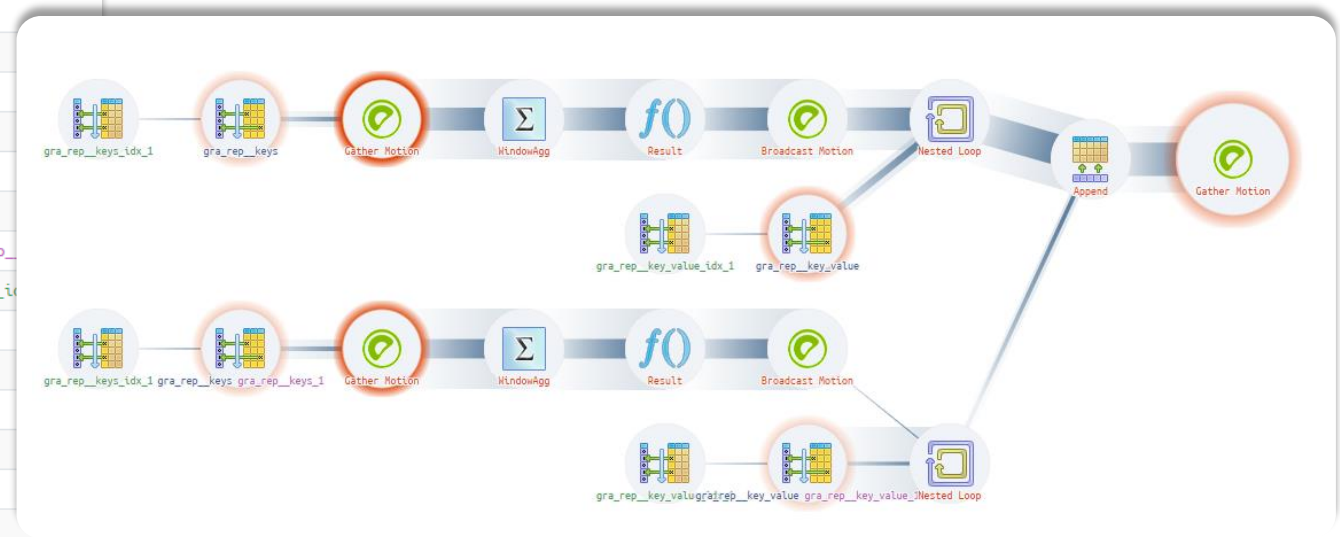


Кейс 5. Greenplum “union all” vs “in”, “or”

План запроса с “union all”. Вариант с явным обращением к таблице с ключами, есть нужные индексы.

#	node, ms	tree, ms	rows	Gather Motion	Gather Motion	Bitmap I	Bitmap Heap S	Gather Motion	Bitmap I	Bitmap
		3.204	2	final results (16B = rows=2 x width=8)						
0	.566	3.204	2 ▼	Gather Motion						
1	-0.118	2.638	1 ▼	-> Append						
2	.037	2.430	1 ▼	-> Nested Loop						
3	.006	1.832	1 ▼	-> Broadcast Motion						
4	.004	1.826	1	-> Result						
5	.013	1.822	1	-> WindowAgg						
6	1.442	1.809	1	-> Gather Motion						
7	.357	.367	1	-> Bitmap Heap Scan on gra_rep__keys						
8	.010		1	-> Bitmap Index Scan on gra_rep__keys_idx_1						
9	.551	.561	1 ▼	-> Bitmap Heap Scan on gra_rep__key_value						
10	.010		1	-> Bitmap Index Scan on gra_rep__key_value_idx_1						
11	.027	.326	1 ▼ CC	-> Nested Loop						
12	-1.542	.005	1 ▼	-> Broadcast Motion						
13	.003	1.547	1	-> Result						
14	.020	1.544	1	-> WindowAgg						
15	1.157	1.524	1	-> Gather Motion						
16	.355	.367	1	-> Bitmap Heap Scan on gra_rep__keys gra_rep_						
17	.012		1	-> Bitmap Index Scan on gra_rep__keys_i						
18	.288	.294	1 ▼	-> Bitmap Heap Scan on gra_rep__key_value gra_rep__key_value_1						
19	.006		1	-> Bitmap Index Scan on gra_rep__key_value_idx_1						
	14.937		EP	Planning time						
				Memory used						
				Optimizer						
	5.557	8.761		Execution time						

4



Другие проблемы PostgreSQL/GP для DWH

1. **Поколончатое хранение**. Производительность, ввод-вывод
2. **Нет пакетного режима (SIMD).** Производительность
3. **Длина имён**. Ограничение. Проблемы при переносе кода
4. **Жёсткий Scheme Binding**. Ограничение применения view
5. **Data Lineage**. Нужен парсинг функций для трекинга зависимостей
6. **Join Elimination**. Производительность
7. **Автогенерация DDL view**. Неудобство применения view

Другие проблемы PostgreSQL/GP для DWH

1. **Поколончатое хранение**. Поддерживается исключительно позаписный режим хранения.

2. **RBAR: ‘Row By Agonizing Row’.** Отсутствие физических операторов плана запроса, поддерживающих пакетный режим обработки записей. Не поддерживается SIMD.

3. **Длина имён**. В PostgreSQL длина поля 63 байта. При импортозамещении, мигрируя из SQL Server, нужно по максимуму сохранить название. В кириллических названиях длина поля сокращается до 31 символов.

4. **Scheme Binding**. Невозможность произвольного обновления представления при наличии зависимых объектов. Новое поле можно добавить только в конец.

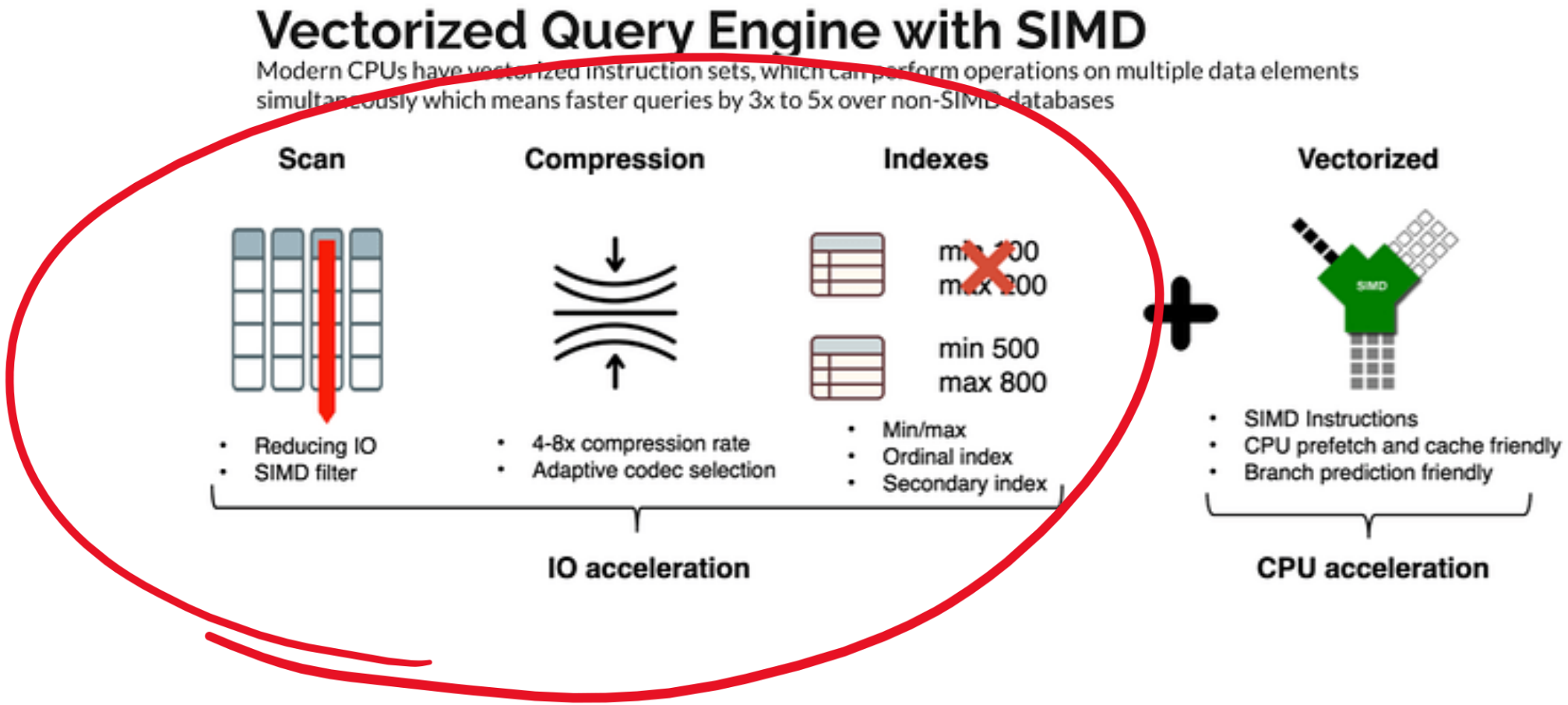
5. **Data Lineage**. Затруднено построение DL для процедур и функций: отсутствует автоматическая генерация зависимостей на языках SQL и PL/pgSQL

6. **Join Elimination**. Если у сущности много сателлитов, то генерируемый запрос содержит много join-ов. Оптимизатор удаляет неиспользуемые join-ы только при использовании PIT-таблиц.

7. **Автогенерация DDL view**. Исходный код, форматирование и комментарии инструкций create or replace view не сохраняются. Инструкция каждый раз генерируется, появляются вызовы функций конвертации типов.

1. Поддержка поколончатого хранения

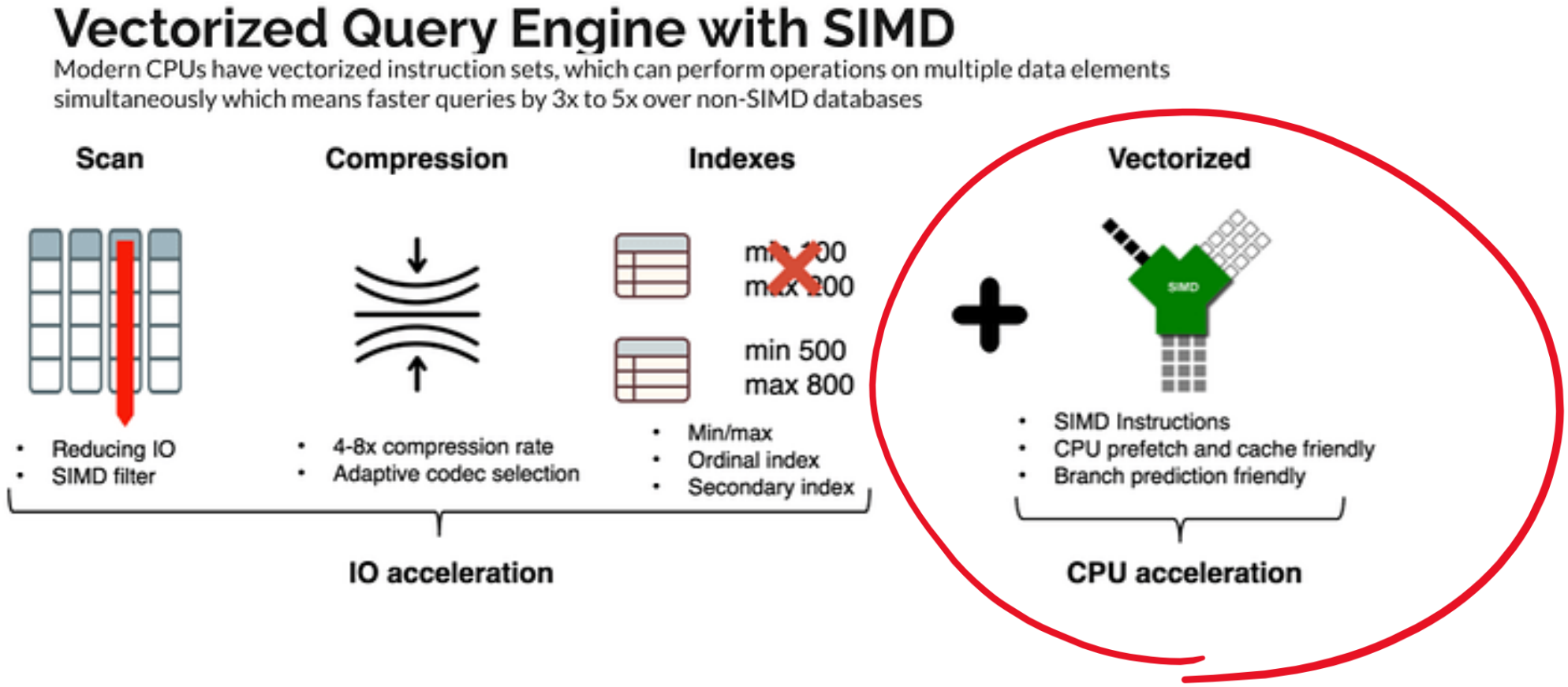
Поддерживается исключительно позаписный режим хранения.



Ожидается решение в Postgres Pro

2. Поддержка SIMD (Single Instruction Multiple Data)

Отсутствие физических операторов плана запроса, поддерживающих пакетный режим обработки записей. Слабая поддержка SIMD.



Ожидается решение в Postgres Pro

3. Длина имён. Настройка в PostgreSQL

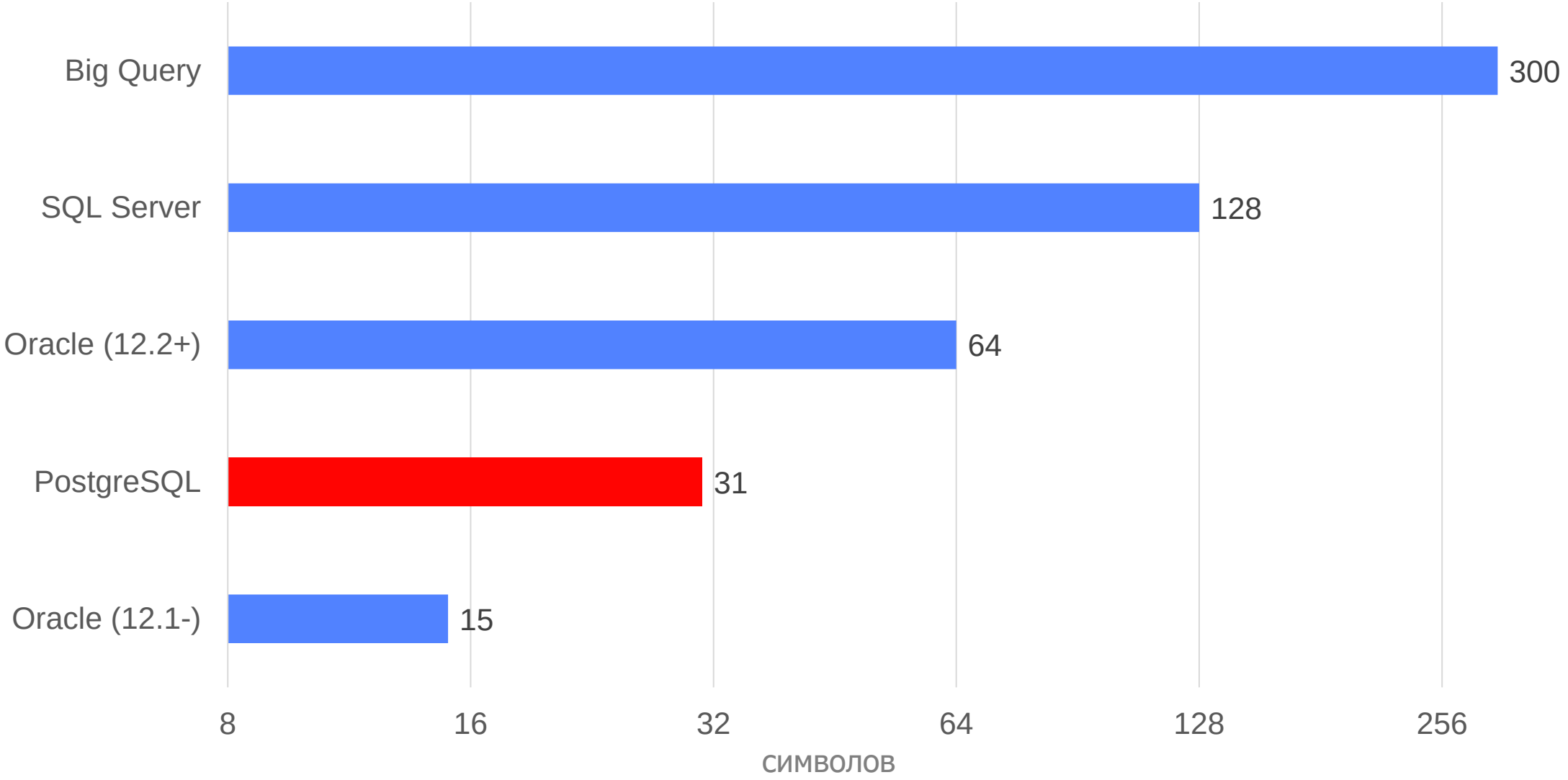
Длина названия поля. В PostgreSQL длина поля 63 байта. При импортозамещении, мигрируя из SQL Server/Oracle, нужно по максимуму сохранить название. В кириллических названиях длина поля сокращается до 31 символов. Изменение на 512 даст 255 символов (вместо текущих 31-го).

postgres / src / include / pg_config_manual.h

```
22      /*
23       * Maximum length for identifiers (e.g. table names, column names,
24       * function names).  Names actually are limited to one fewer byte than this,
25       * because the length must include a trailing zero byte.
26       *
27       * Changing this requires an initdb.
28       */
29      #define NAMEDATALEN 64
30
```

Есть workaround в BI.Qube

3. Длина имён. Max в 2-х байтной кодировке



Есть workaround в BI.Qube

4. Жёсткий Scheme Binding представлений

Блокировка обновления представления при наличии зависимостей

Нельзя: менять тип данных полей, удалять, добавлять поле (не в конец)

```
create view view_1 as select a, b from table
```

зависимость

```
    create view view_2 as select * from view_1
    ...
```

новое поле не в конец

```
create or replace view view_1 as select a, c, b from table
```

падает из-за зависимостей

```
    create view view_name with no schema binding as select...
```

необходим режим

Есть workaround в BI.Qube

5. Отсутствие трекинга зависимостей функций sql

Затруднено построение Data Llineage для процедур и функций:
нет автоматической генерация зависимостей на языках SQL и PL/pgSQL

Рекомендация:

Использовать сторонние парсеры,
например расширение plpgsql_check

Есть workaround в BI.Qube

6. Join Elimination – не работает на всех кейсах

Если у сущности много сателлитов, то генерируемый запрос содержит много join-ов

Планировщик запросов удаляет неиспользуемые join-ы только при использовании PIT-таблиц

lateral join + limit 1

```
left join lateral
(
   select *
   from vault.sat_entity_1
   where entity_id = ent.id
   order by
      load_date desc
      limit 1
) sat_1 on true
```

флаг последней записи

```
left join vault.sat_entity_1 sat_1 on
   sat_1.entity_id = ent.id
   and sat_1.is_last_record


create unique index sat_entity_1_last_only
on vault.sat_entity_1 (entity_id)
where is_last_record;
```

Есть workaround в BI.Qube

7. Автогенерация DDL view

Исходный код, форматирование и комментарии ddl “create or replace view” не сохраняются.

Инструкция каждый раз генерируется, появляются вызовы функций конвертации типов.

Рекомендация:

Вместо автогенерации кода представлений непосредственно из базы данных использовать source control

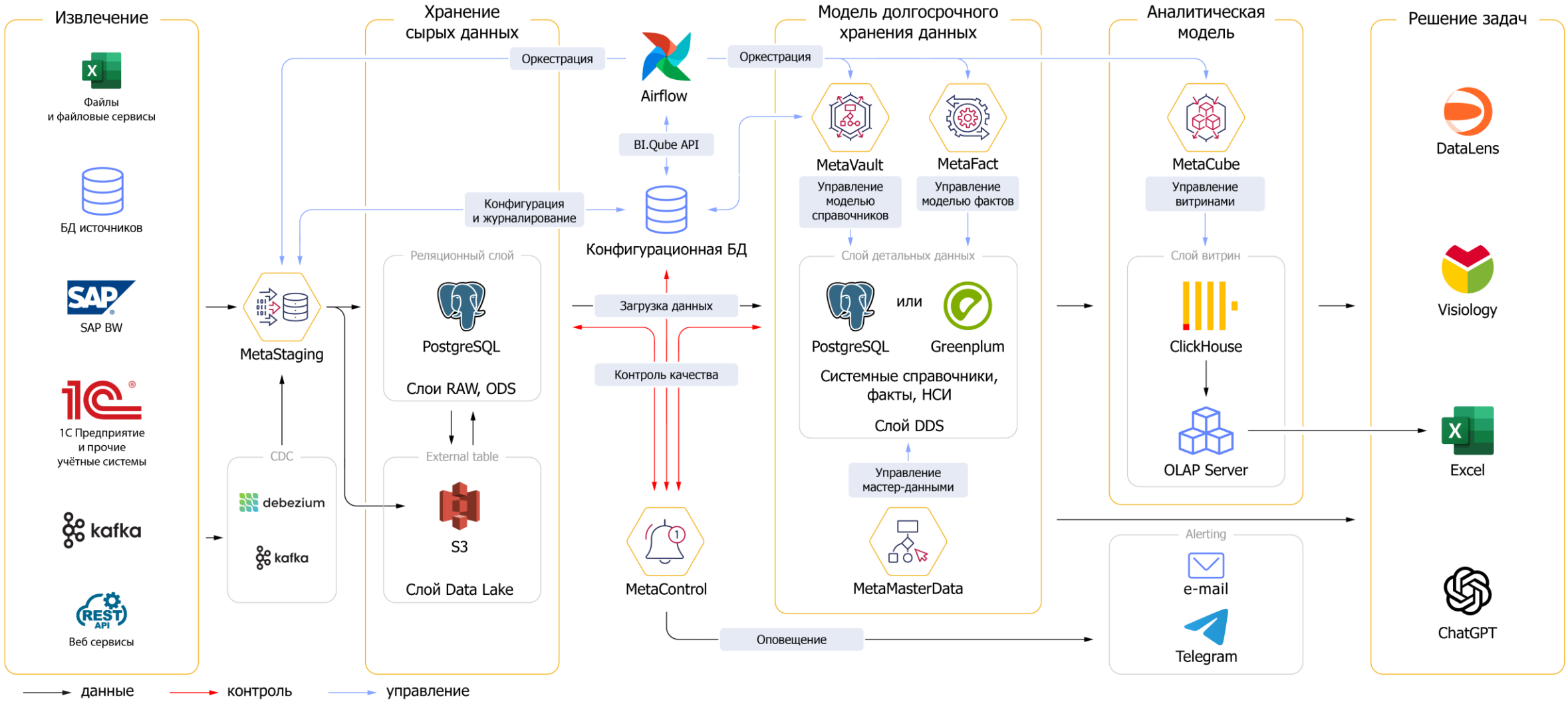
Есть workaround

Автоматизация разработки с применением фреймворка BI.Qube

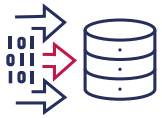
Доступно on-premises и в Yandex Cloud

BI.Qube

Универсальная архитектура DWH



BI.Qube 2.0 — фреймворк нового поколения



MetaStaging

Для интеграции данных из СУБД, 1С, API, web-сервисов

В Реестре российского ПО (digital.gov.ru)

MetaVault

Для построения единой масштабируемой модели DWH

В Реестре российского ПО (digital.gov.ru)

MetaControl

Для обеспечения качества данных и оповещений

В Реестре российского ПО (digital.gov.ru)

MetaMasterData

Для управления основными данными компании

Preview

MetaFact

Для low-code формирования фактов

Alpha

MetaCube

Для переноса модели данных в витрины

Преимущества фреймворка для заказчика

Ускорение проектов

- автоматизация развертывания
- быстрое наполнение DWH
- короткий time-to-market

Быстрый старт

- простая подписка на продукт из платежного аккаунта Yandex Cloud
- переход к использованию продуктов open source в виде управляемых сервисов Yandex Cloud Data Platform (Object Storage, PostgreSQL, Greenplum, ClickHouse, DataLens)
- консалтинг по подписке

Снижение требований к экспертизе

- генерация ELT по метаданным
- low-code/no-code для автоматизации типовых задач DWH
- сквозной Data Governance

Повышение эффективности

- ускорение разработки в 2-5 раз
- сокращение трудозатрат
- минимизация ошибок в коде
- уменьшение лицензионных платежей
- снижение TCO за счет упрощения поддержки и внесения изменений

Выводы

Для проектов DWH критично автоматизировать:

- **Моделирование DWH** (Data Vault, включая Business Vault)
- **Data Governance** (Data Linage, Data Quality)
- **Master Data Management** (ведение НСИ, дедубликация, золотая запись)
- **CI/CD**

Разработка DWH с использованием правильных инструментов позволяет:

- **Сократить время** на разработку проекта
- **Снизить затраты** на поддержку и развитие DWH
- Обеспечить **высокий уровень** качества полученного продукта
- **Снизить порог входа** и требования к компетенциям для разработчиков

Благодарим за внимание

Фреймворк
BI.Qube

BI.Qube в Маркетплейсе
Yandex Cloud

BI.Qube