Как объединять данные из разных СУБД и делать это эффективно

Виталий Исаев

Яндекс, YDB

HL 2024 HighLoad++

Объединение данных

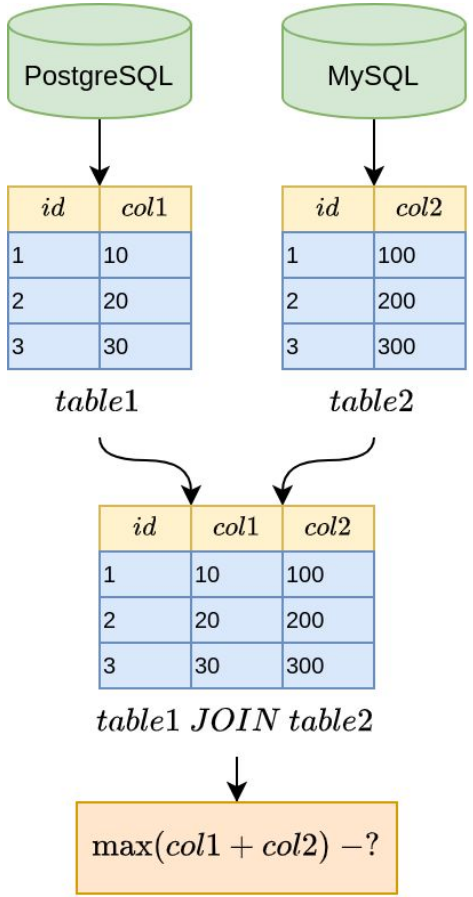
?

Автор

- МойОфис: объектное хранилище Mailion
- Яндекс: YDB, Yandex Query
- Go / C++ / Python

JOIN таблиц из разных баз

- Две разные базы данных
- В каждой базе по таблице
- В каждой таблице есть PK
- Нужно объединить таблицы и вычислить максимум суммы столбцов



Наивное решение


```
import pandas as pd
import sqlalchemy
```

Наивное решение


```
import pandas as pd
import sqlalchemy

pg = sqlalchemy.create_engine("postgresql://localhost:5432/db")
df1 = pd.read_sql("SELECT * FROM table", pg, index_col="id")
```

Наивное решение


```
import pandas as pd
import sqlalchemy

pg = sqlalchemy.create_engine("postgresql://localhost:5432/db")
df1 = pd.read_sql("SELECT * FROM table", pg, index_col="id")

my = sqlalchemy.create_engine("mysql://localhost:3306/db")
df2 = pd.read_sql("SELECT * FROM table", my, index_col="id")
```

Наивное решение


```
import pandas as pd
import sqlalchemy

pg = sqlalchemy.create_engine("postgresql://localhost:5432/db")
df1 = pd.read_sql("SELECT * FROM table", pg, index_col="id")

my = sqlalchemy.create_engine("mysql://localhost:3306/db")
df2 = pd.read_sql("SELECT * FROM table", my, index_col="id")

df = df1.join(df2)
```

Наивное решение


```
import pandas as pd
import sqlalchemy

pg = sqlalchemy.create_engine("postgresql://localhost:5432/db")
df1 = pd.read_sql("SELECT * FROM table", pg, index_col="id")

my = sqlalchemy.create_engine("mysql://localhost:3306/db")
df2 = pd.read_sql("SELECT * FROM table", my, index_col="id")

df = df1.join(df2)

result = df[["col1", "col2"]].sum(axis=1).max()
```

Наивное решение: минусы

- Высокое потребление оперативной памяти.

Наивное решение: минусы

- Высокое потребление оперативной памяти.
- Сложности параллельной обработки.

Наивное решение: минусы

- Высокое потребление оперативной памяти.
- Сложности параллельной обработки.
- Низкая выразительность ЯП.

Выразительность языка

Декларативный язык:

```
SELECT
  MAX(t1.col1 + t2.col2)
FROM
  pg.table1 AS t1
  JOIN
  my.table2 AS t2
ON t1.id = t2.id;
```

Чего нужно достигнуть?

Выразительность языка

Декларативный язык:

```
SELECT
  MAX(t1.col1 + t2.col2)
FROM
  pg.table1 AS t1
  JOIN
  my.table2 AS t2
ON t1.id = t2.id;
```

Чего нужно достигнуть?

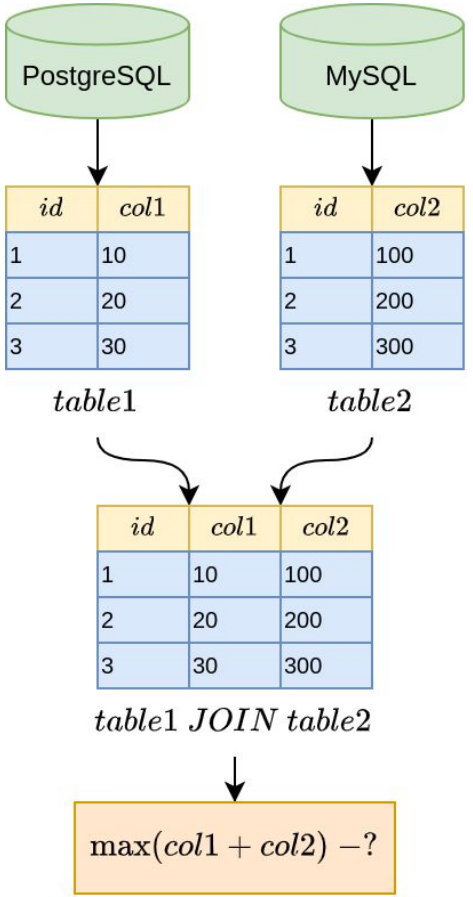
Императивный язык:

```
df1 = pd.read_sql(
    "SELECT * FROM table", pg,
    index_col="id")

df2 = pd.read_sql(
    "SELECT * FROM table", my,
    index_col="id")

df = df1.join(df2)

result = df[["col1", "col2"]].
    sum(axis=1).max()
```

Как этого достигнуть?

Два чтения в одном запросе


```
SELECT
  MAX(t1.col1 + t2.col2)
FROM
  pg.table1 AS t1
  JOIN
  my.table2 AS t2
ON t1.id = t2.id;
```



Федеративные SQL-запросы — запросы, адресованные к внешним источникам данных.

О федеративных базах данных

Федеративность баз данных

- **Федеративная база данных** — совокупность взаимодействующих, но автономных БД.
- **Федеративная СУБД** — связующий слой ПО между автономными базами.

Федеративность баз данных

- **Федеративная база данных** — совокупность взаимодействующих, но автономных БД.
- **Федеративная СУБД** — связующий слой ПО между автономными базами.

Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases[1]

AMIT P. SHETH

Bellcore, 1J-210, 444 Hoes Lane, Piscataway, New Jersey 08854

JAMES A. LARSON

Intel Corp., HF3-02, 5200 NE Elam Young Pkwy., Hillsboro, Oregon 97124

A federated database system (FDBS) is a collection of cooperating database systems that are autonomous and possibly heterogeneous. In this paper, we define a reference architecture for distributed database management systems from system and schema viewpoints and show how various FDBS architectures can be developed. We then define a methodology for developing one of the popular architectures of an FDBS. Finally, we discuss critical issues related to developing and operating an FDBS.

Categories and Subject Descriptors: D.2.1 [**Software Engineering**]: Requirements/Specifications—*methodologies*; D.2.10 [**Software Engineering**]: Design; H.0 [**Information Systems**]: General; H.2.0 [**Database Management**]: General; H.2.1 [**Database Management**]: Logical Design—*data models, schema and subschema*; H.2.4 [**Database Management**]: Systems; H.2.5 [**Database Management**]: Heterogeneous Databases; H.2.7 [**Database Management**]: Database Administration

General Terms: Design, Management

Additional Key Words and Phrases: Access control, database administrator, database design and integration, distributed DBMS, federated database system, heterogeneous DBMS, multidatabase language, negotiation, operation transformation, query processing and optimization, reference architecture, schema integration, schema translation, system evolution methodology, system/schema/processor architecture, transaction management

Классификация

Системы с федеративными возможностями:

Классификация

Системы с федеративными возможностями:

- Транзакционные (OLTP) СУБД

Классификация

Системы с федеративными возможностями:

- Транзакционные (OLTP) СУБД
- Аналитические (OLAP) СУБД

Классификация

Системы с федеративными возможностями:

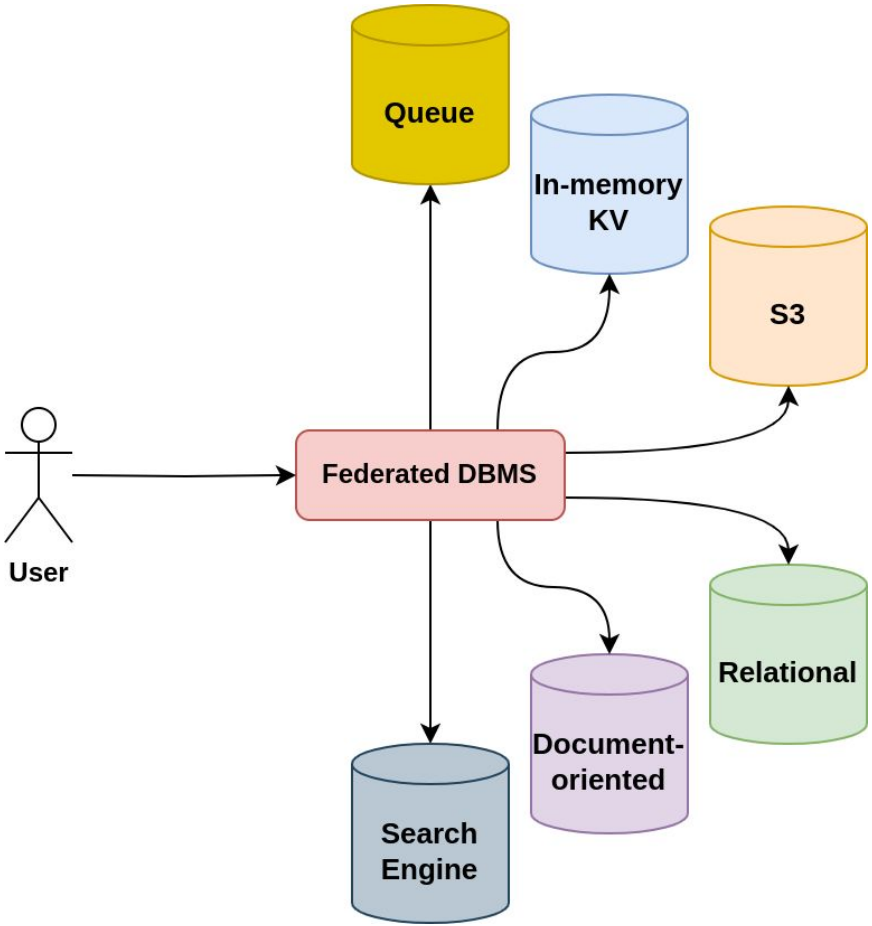
- Транзакционные (OLTP) СУБД
- Аналитические (OLAP) СУБД
- Движки обработки запросов

Классификация

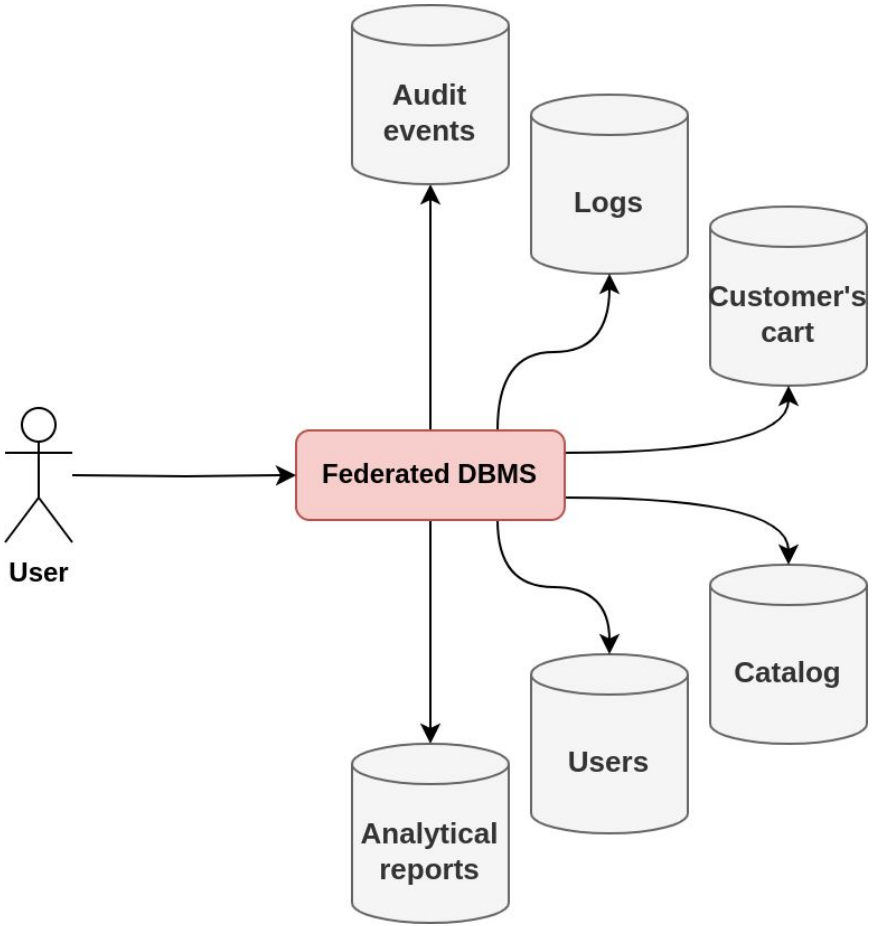
Системы с федеративными возможностями:

- Транзакционные (OLTP) СУБД
- Аналитические (OLAP) СУБД
- Движки обработки запросов

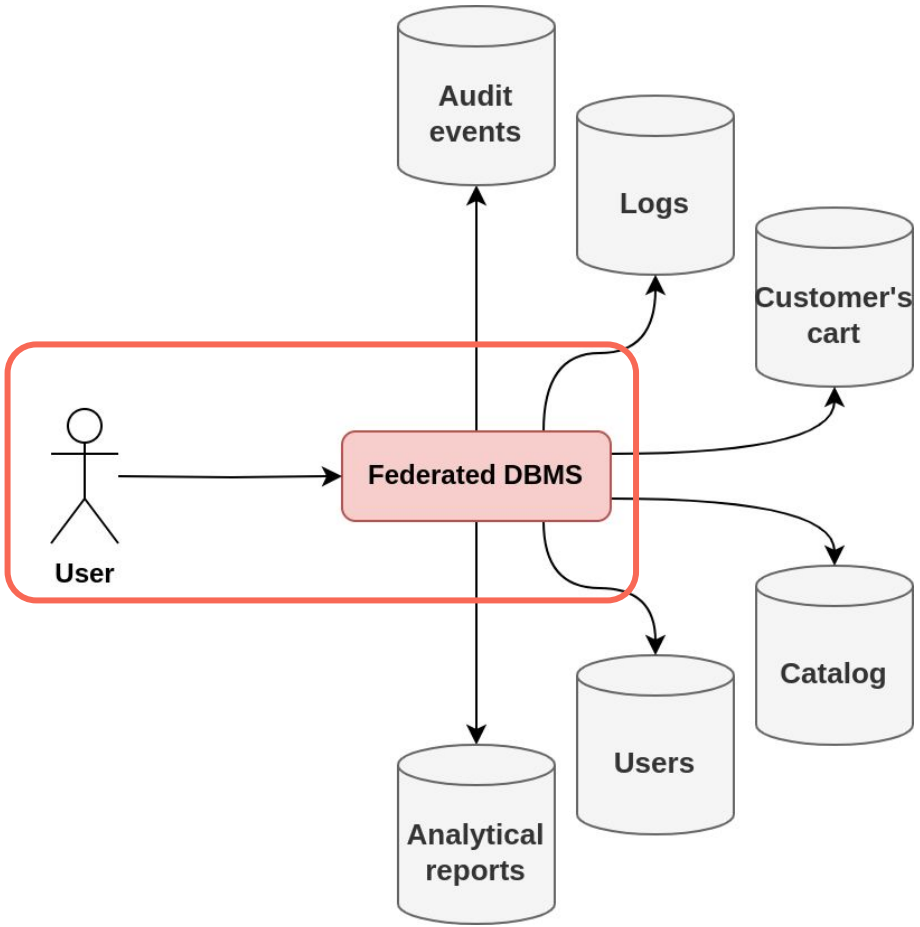
Разнотипные источники данных



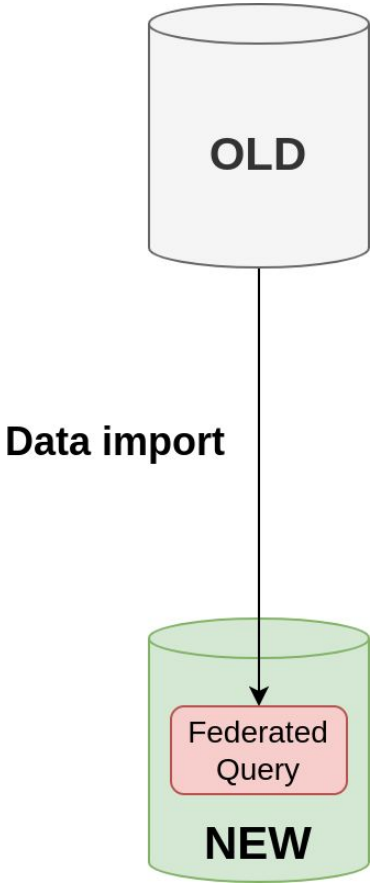
Однотипные источники данных



Однотипные источники данных

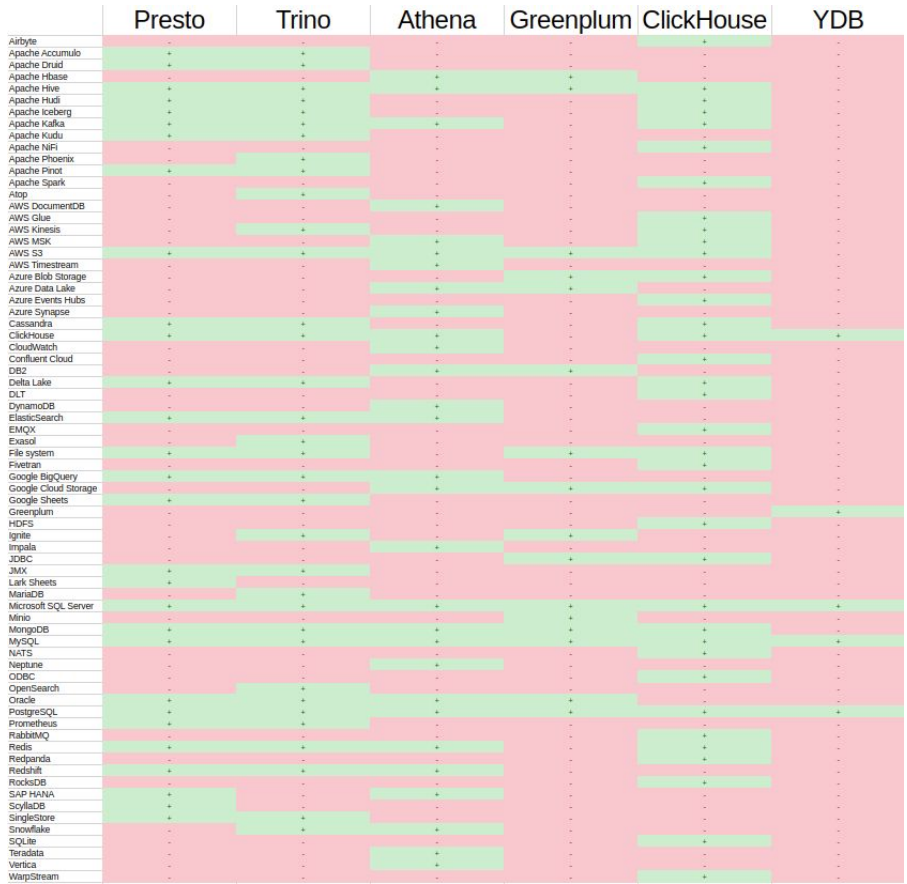


Миграция данных



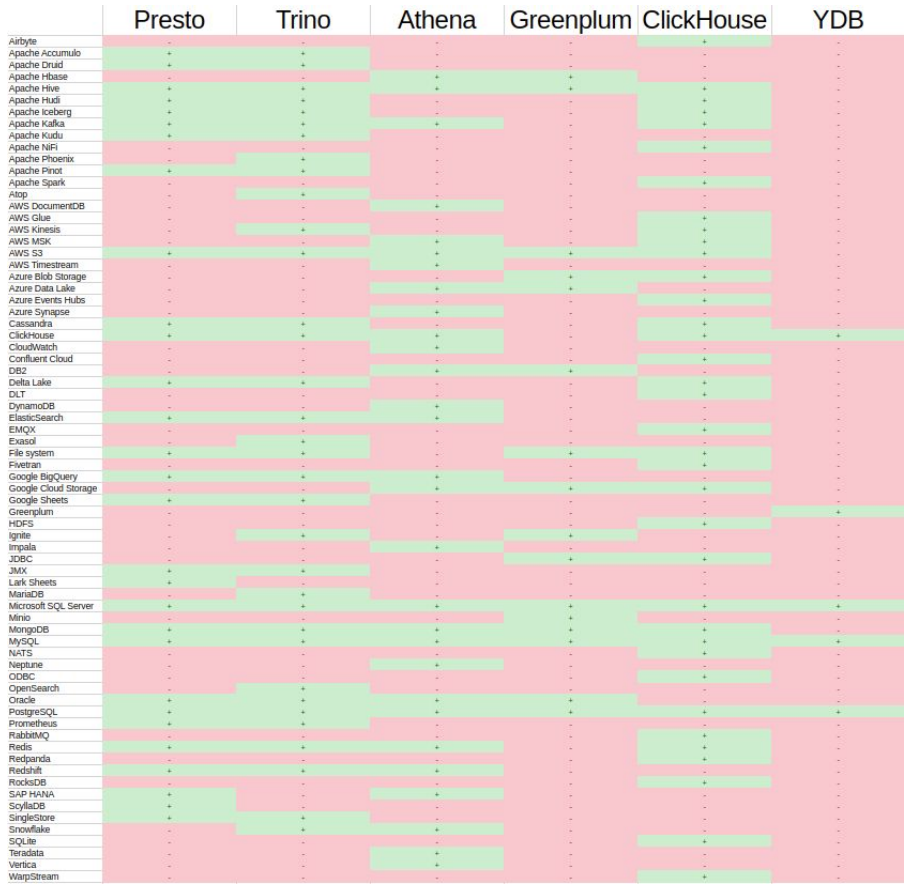
Широта охвата

- Чем больше интеграций, тем лучше (у лидеров > 30 источников).



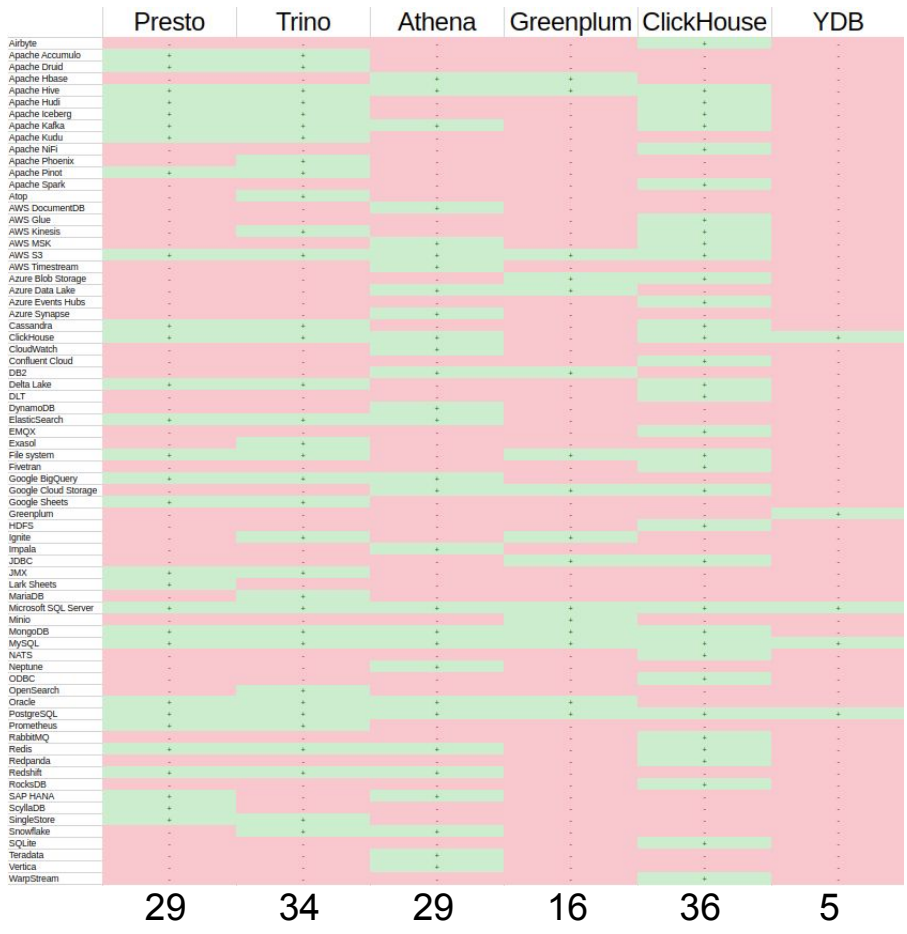
Широта охвата

- Чем больше интеграций, тем лучше (у лидеров > 30 источников).
- Источники **очень сильно отличаются.**

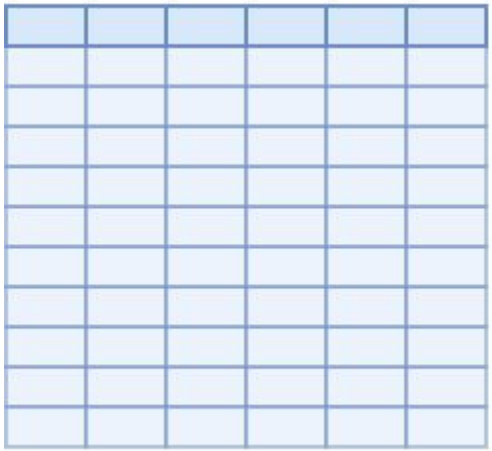


Широта охвата

- Чем больше интеграций, тем лучше (у лидеров > 30 источников).
- Источники **очень сильно отличаются.**
- Гетерогенность источников ⇒ трудность построения универсальной FDBMS.



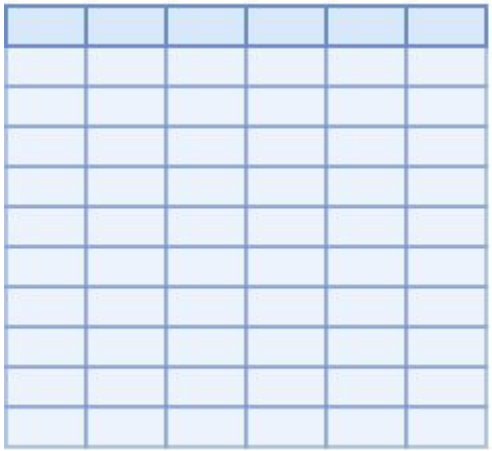
Разные модели данных

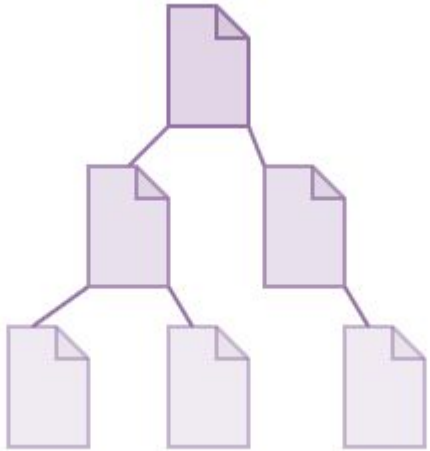


Relational:

- RDBMS
- Columnar
- Spreadsheets
- S3: CSV, Parquet

Разные модели данных





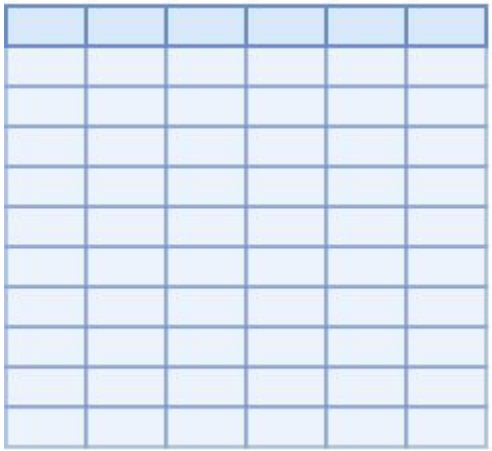
Relational:

- RDBMS
- Columnar
- Spreadsheets
- S3: CSV, Parquet

Document:

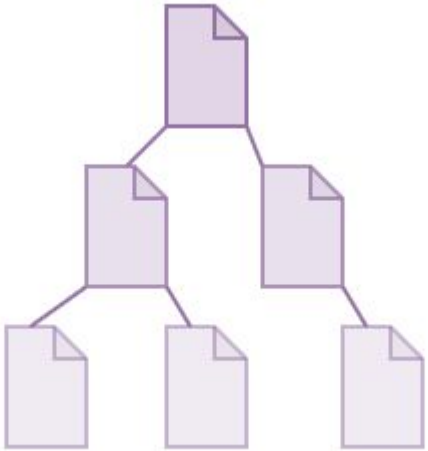
- NoSQL
- Search indexes
- S3: JSON

Разные модели данных



Relational:

- RDBMS
- Columnar
- Spreadsheets
- S3: CSV, Parquet



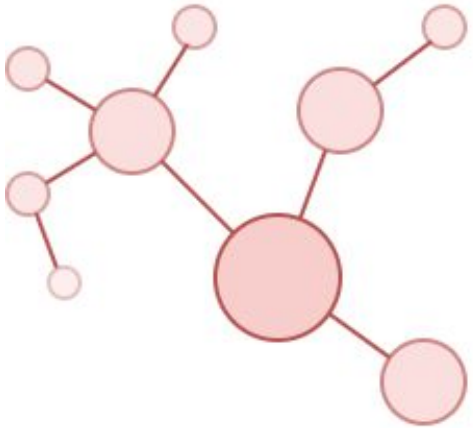
Document:

- NoSQL
- Search indexes
- S3: JSON

Key-value:

- NoSQL
- In-memory KV
- In-memory Data Grids

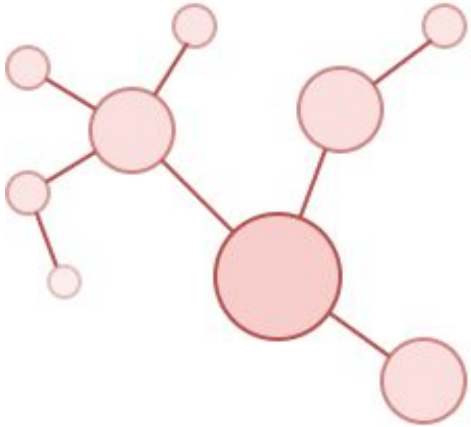
Разные модели данных

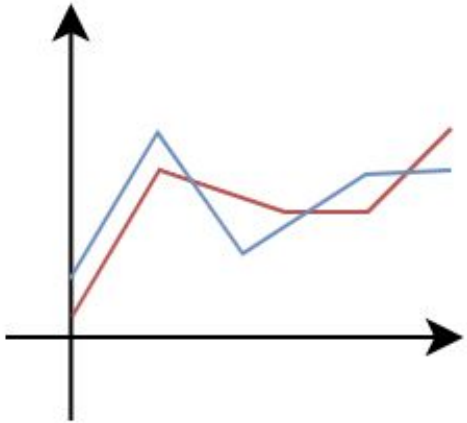


Network:

- Graph databases

Разные модели данных





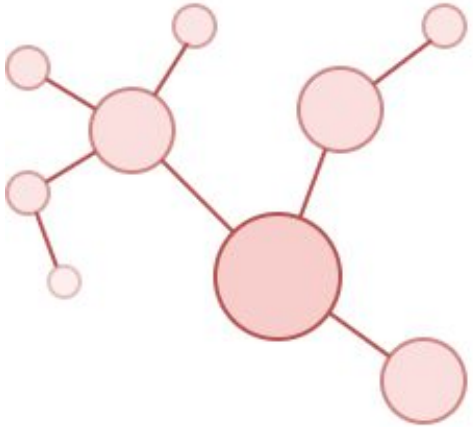
Network:

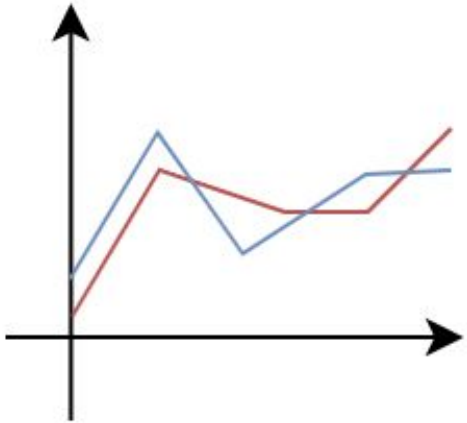
- Graph databases

Time series:

- Monitoring

Разные модели данных





Network:

- Graph databases

Time series:

- Monitoring

Streams:

- Message queues
- Streaming platforms

Разные интерфейсы

Разные интерфейсы

RDBMS (SQL):

```
SELECT * FROM users
    WHERE age > 25
```

Разные интерфейсы

RDBMS (SQL):

```
SELECT * FROM users
    WHERE age > 25
```

MongoDB (MQL):

```
db.users.find(
    {age: { $gt: 25 } }
)
```

Разные интерфейсы

RDBMS (SQL):

```
SELECT * FROM users
    WHERE age > 25
```

Dgraph (GraphQL):

```
query {
  allUsers(filter: { age: { gt: 25 } }) {
    data {id name age}
}}
```

MongoDB (MQL):

```
db.users.find(
    {age: { $gt: 25 } }
)
```

Разные интерфейсы

RDBMS (SQL):

```
SELECT * FROM users
    WHERE age > 25
```

Dgraph (GraphQL):

```
query {
  allUsers(filter: { age: { gt: 25 } }) {
    data {id name age}
}}
```

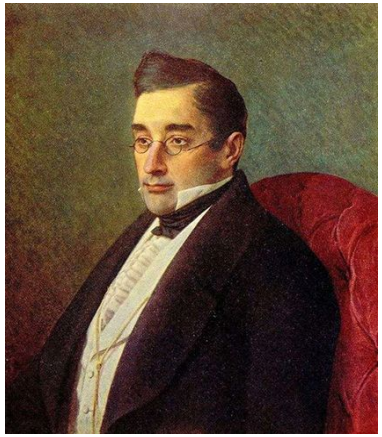
MongoDB (MQL):

```
db.users.find(
    {age: { $gt: 25 } }
)
```

S3:

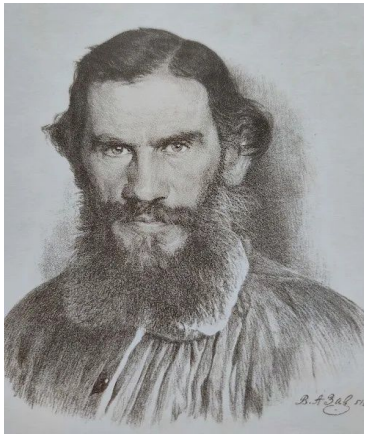
```
GET /users.csv?select&select-type=2 HTTP/1.1
Host: bucket.storage.yandexcloud.net
{
  "Expression":
    "SELECT * FROM Obj WHERE age > 25"
}
```

Полиглоты

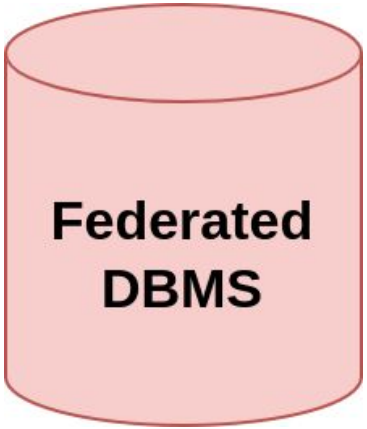


10

14



15



∞

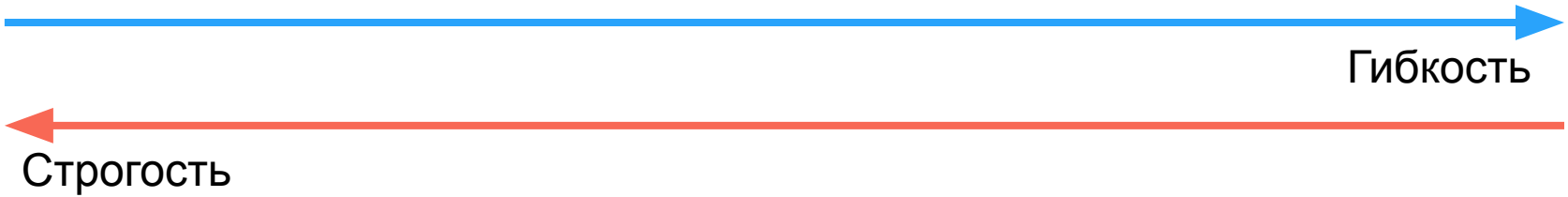
Разные подходы к схематизации

Схема БД = таблицы + колонки + связи

Схема внешнего источника данных **требуется** для обработки федеративного запроса

Разные подходы к схематизации

Schema-on-write



Разные подходы к схематизации

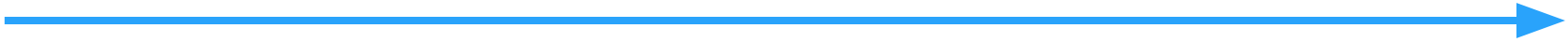
Schema-on-write

PostgreSQL

ClickHouse

YDB

Schemaless



Гибкость

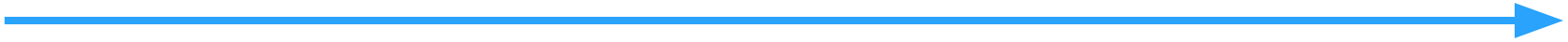
Строгость

Разные подходы к схематизации

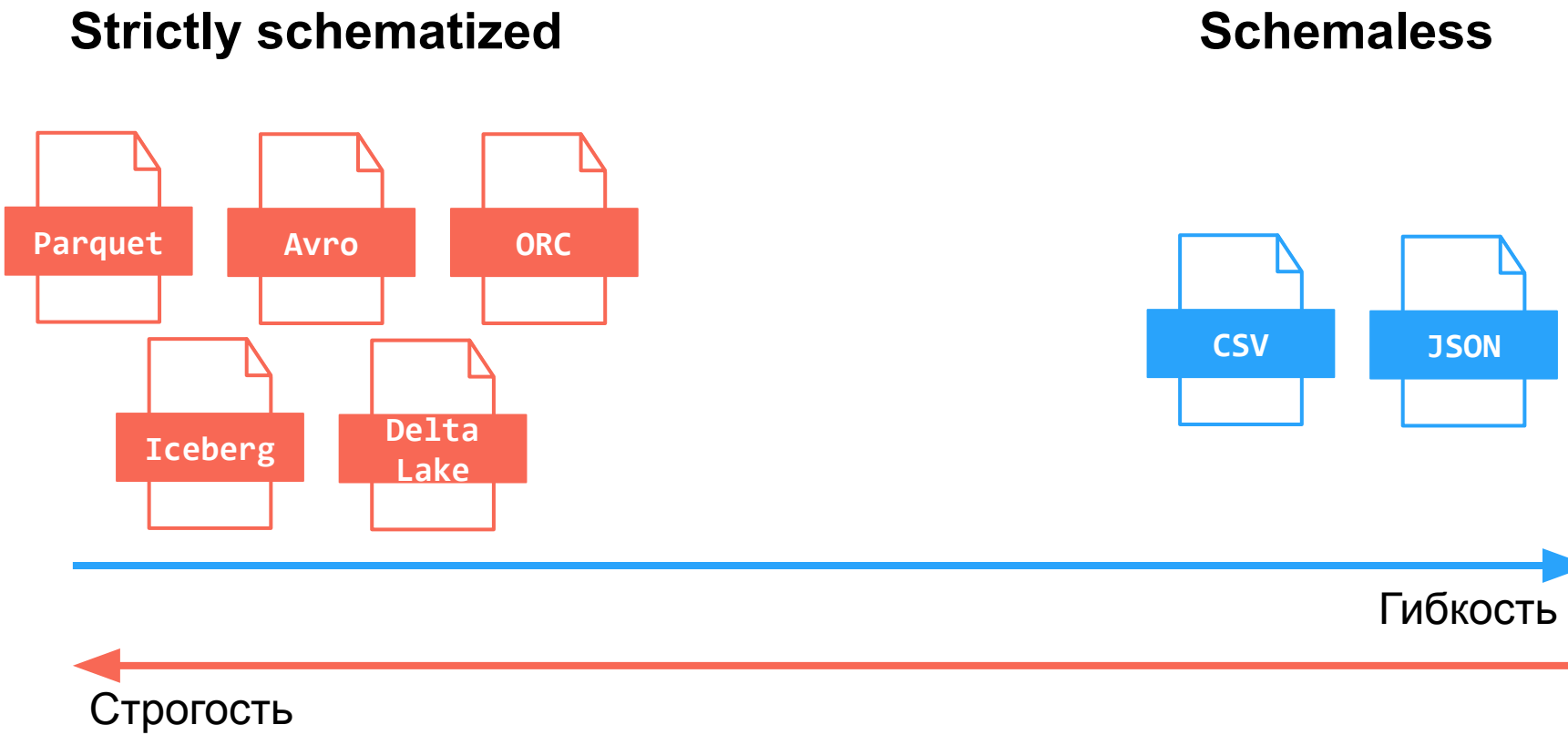
Schema-on-write

Schema-on-read

Schemaless



Форматы данных в S3



Schemaless источники: подходы

Что делать, если у источника данных нет схемы?

Schemaless источники: подходы

Схема предоставляется пользователем

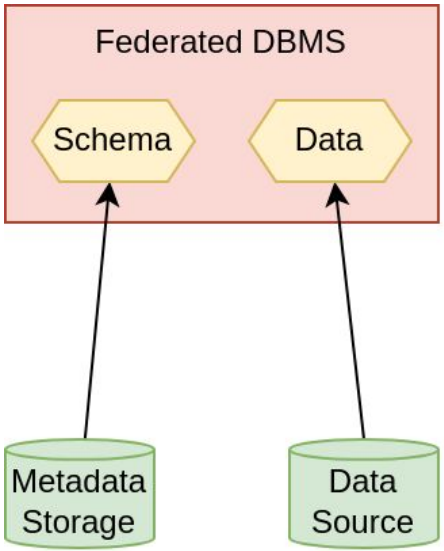
```
SELECT *
FROM users.csv
WITH(
  SCHEMA(
    Id int,
    Name String,
    Age String
  )
);
```

Schemaless источники: подходы

Схема предоставляется пользователем

```
SELECT *
FROM users.csv
WITH(
  SCHEMA(
    Id int,
    Name String,
    Age String
  )
);
```

Схема находится во внешнем хранилище метаданных



Schemaless источники: подходы

Схема предоставляется пользователем

```
SELECT *
FROM users.csv
WITH(
  SCHEMA(
    Id int,
    Name String,
    Age String
  )
);
```

Схема находится во внешнем хранилище метаданных

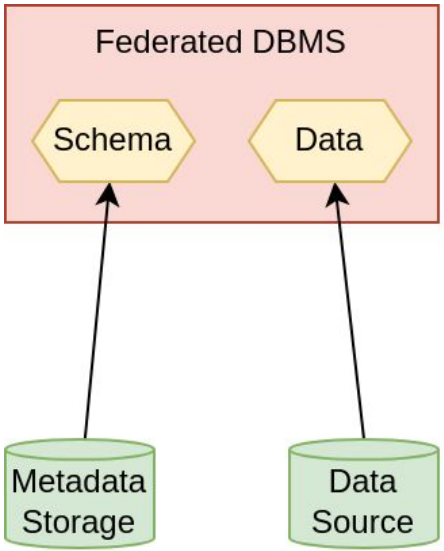
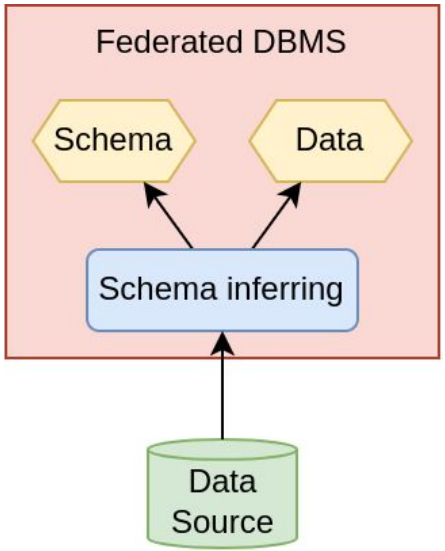


Схема выводится в момент чтения



Разные системы типов

Разные системы типов

MySQL (1995)	**ClickHouse** (2016)	**YDB** (2019)
`tinyint`	`Int8`	`Int8`
`tinyint unsigned`	`UInt8`	`Uint8`
`smallint`	`Int16`	`Int16`
`smallint unsigned`	`UInt16`	`Uint16`
`mediumint`	`Int32`	`Int32`
`mediumint unsigned`	`UInt32`	`Uint32`
`bigint`	`Int64`	`Int64`
`bigint unsigned`	`UInt64`	`Uint64`
`Decimal(P,S)`	`Decimal(P,S)`	`Decimal(22,9)`

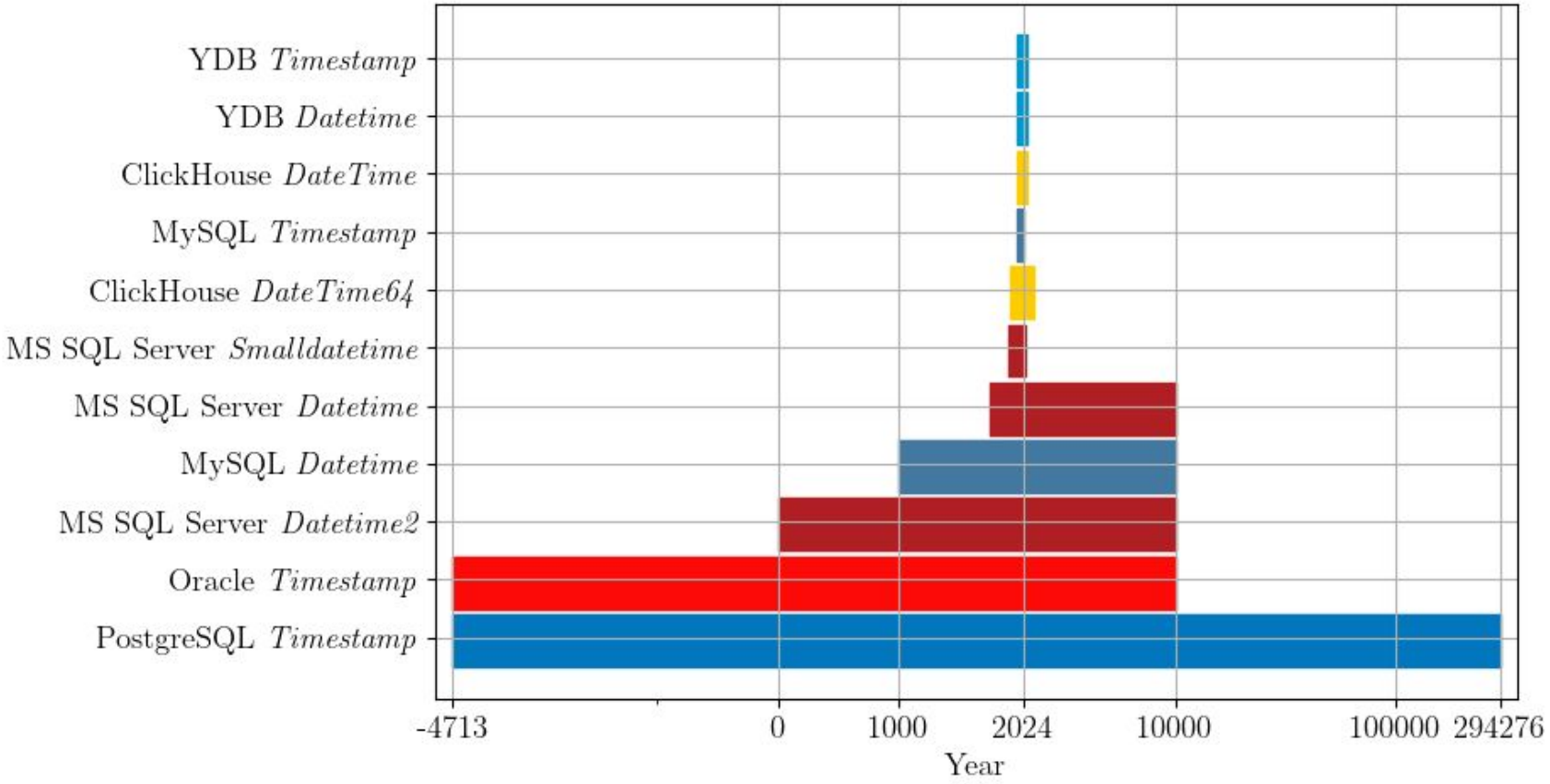
Разные системы типов

Oracle (1976)	**PostgreSQL** (1986)	**SQL Server** (1989)	**MySQL** (1995)	**ClickHouse** (2016)	**YDB** (2019)
		`tinyint`	`tinyint`	`Int8`	`Int8`
			`tinyint unsigned`	`UInt8`	`Uint8`
	`smallint`	`smallint`	`smallint`	`Int16`	`Int16`
			`smallint unsigned`	`UInt16`	`Uint16`
	`integer`	`int`	`mediumint`	`Int32`	`Int32`
			`mediumint unsigned`	`UInt32`	`Uint32`
	`bigint`	`bigint`	`bigint`	`Int64`	`Int64`
			`bigint unsigned`	`UInt64`	`Uint64`
`NUMBER(P,S)`	`numeric(p,s)`	`Decimal(P,S)`	`Decimal(P,S)`	`Decimal(P,S)`	`Decimal(22,9)`

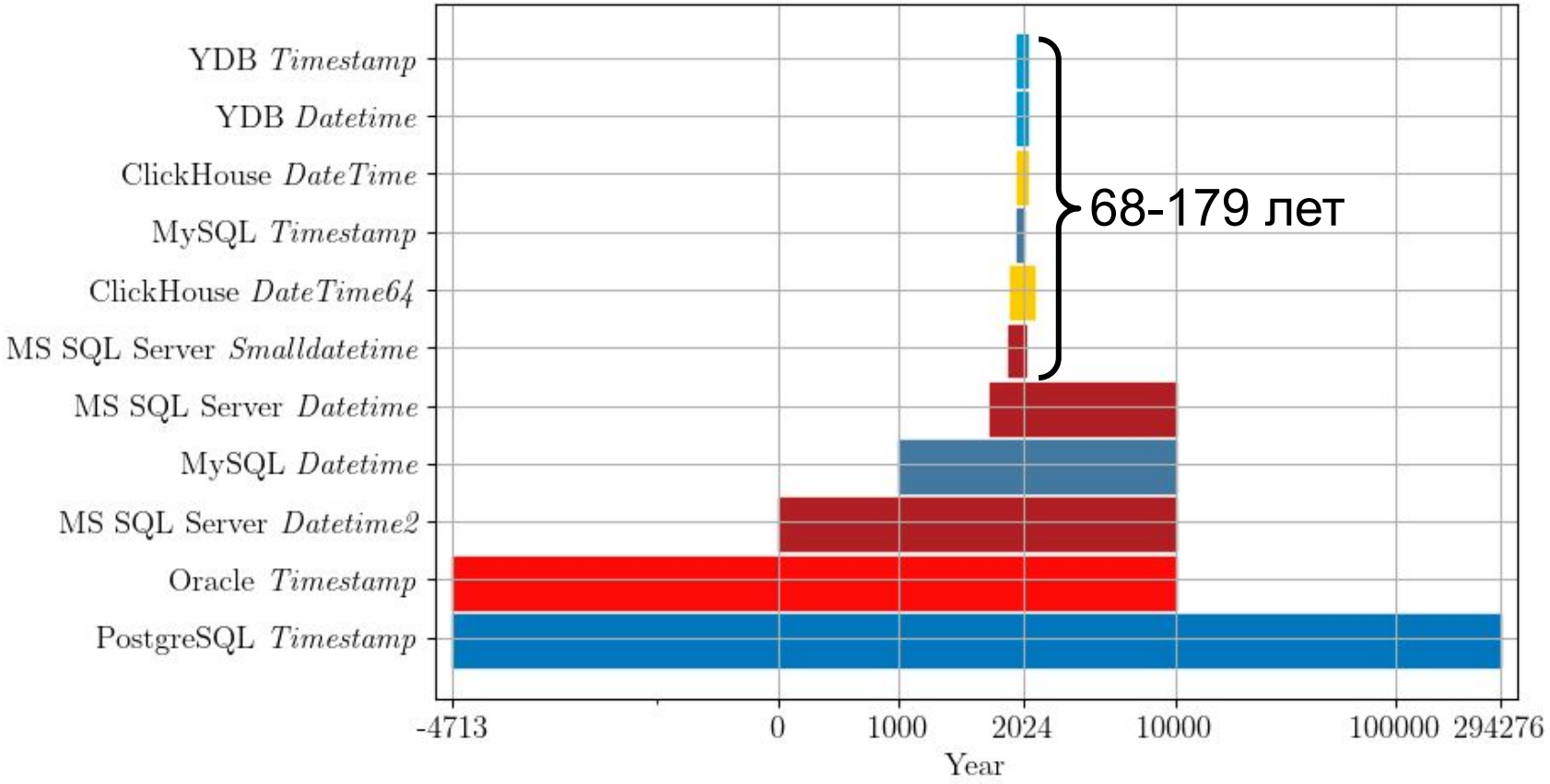
Разные системы типов

Oracle (1976)	**PostgreSQL** (1986)	**SQL Server** (1989)	**MySQL** (1995)	**ClickHouse** (2016)	**YDB** (2019)
		`tinyint`	`tinyint`	`Int8`	`Int8`
			`tinyint unsigned`	`UInt8`	`Uint8`
	`smallint`	`smallint`	`smallint`	`Int16`	`Int16`
			`smallint unsigned`	`UInt16`	`Uint16`
	`integer`	`int`	`mediumint`	`Int32`	`Int32`
			`mediumint unsigned`	`UInt32`	`Uint32`
	`bigint`	`bigint`	`bigint`	`Int64`	`Int64`
от 1 до 21 байт			`bigint unsigned`	`UInt64`	`Uint64`
`NUMBER(P,S)`	`numeric(p,s)`	`Decimal(P,S)`	`Decimal(P,S)`	`Decimal(P,S)`	`Decimal(22,9)`

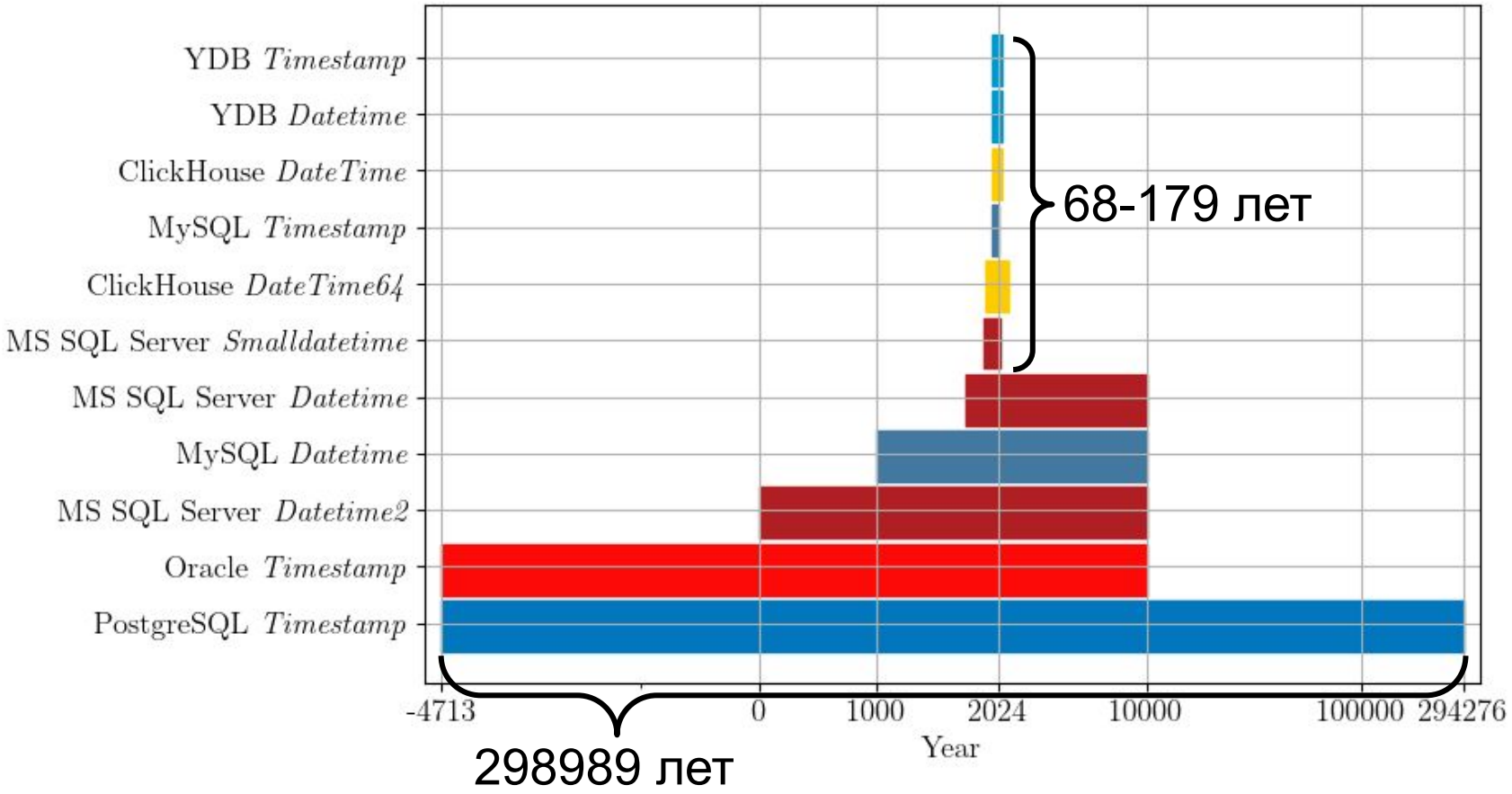
Разные системы типов



Разные системы типов



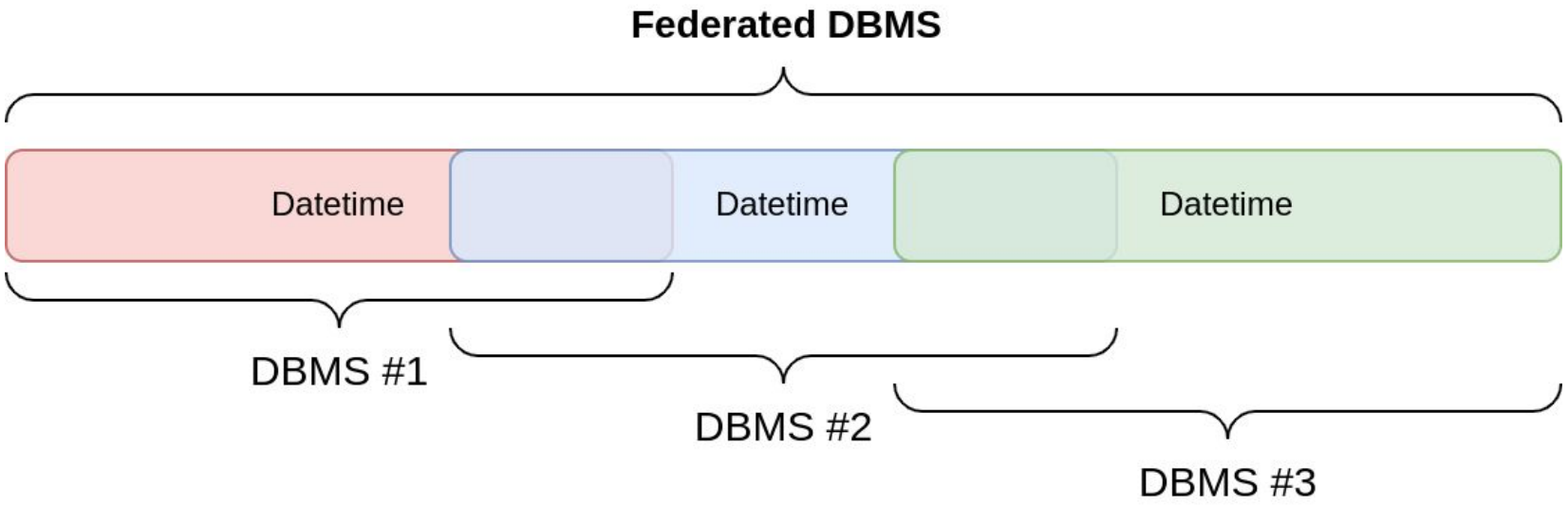
Разные системы типов



Разные системы типов

И это мы ещё точности не касались…

Суперпозиция диапазонов типов

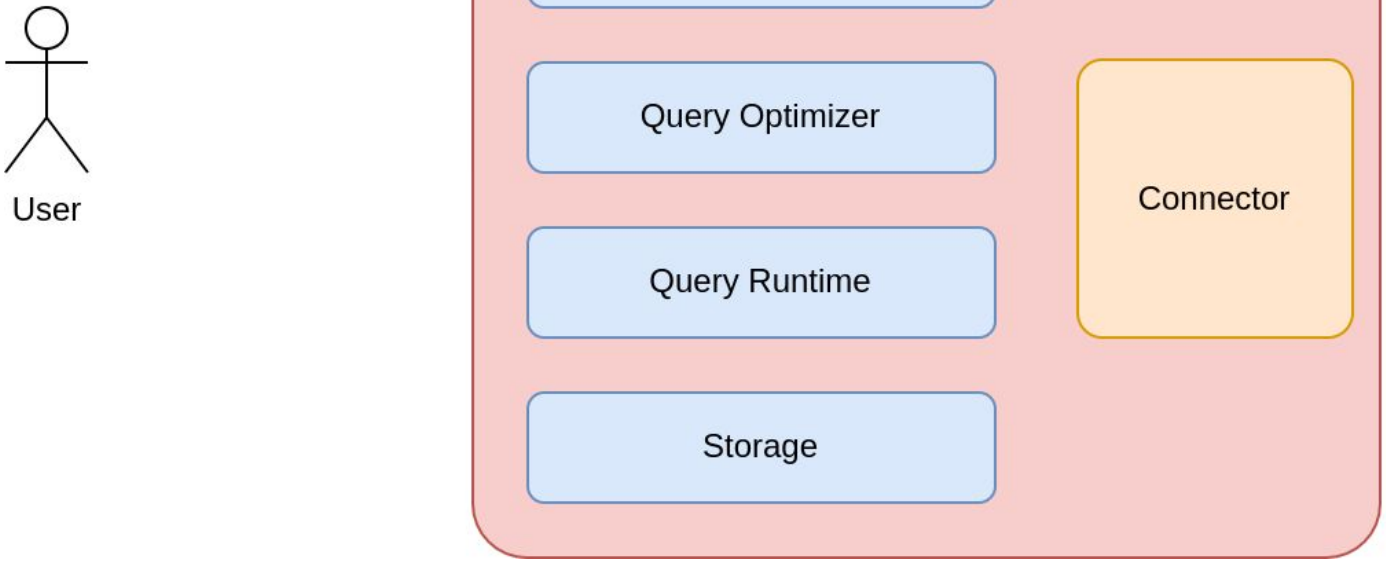


Федеративные системы: итоги

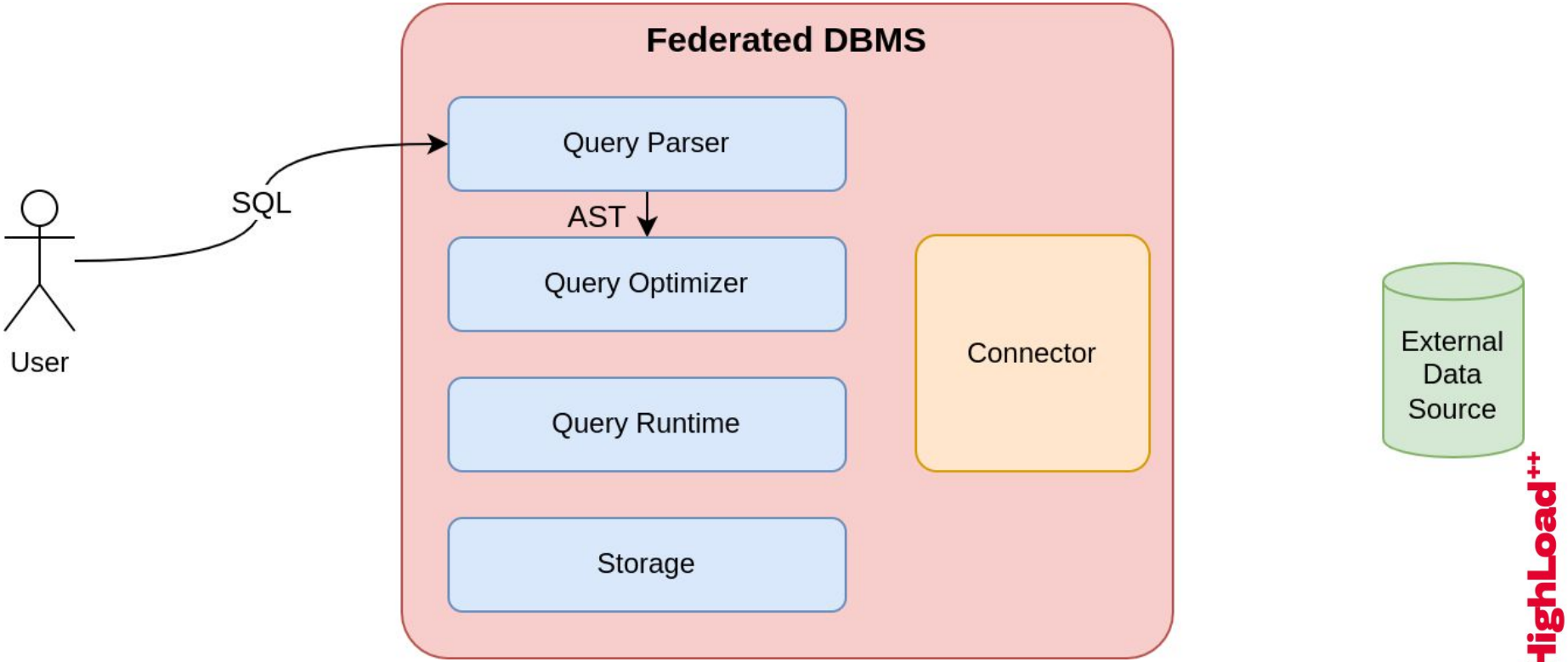
- **Для чего нужны**: объединение и анализ данных из внешних источников, миграция данных.
- **Основные сложности**: гетерогенность моделей данных, интерфейсов, подходов к схематизации и систем типов внешних источников данных.

Как федеративные базы читают из внешнего источника

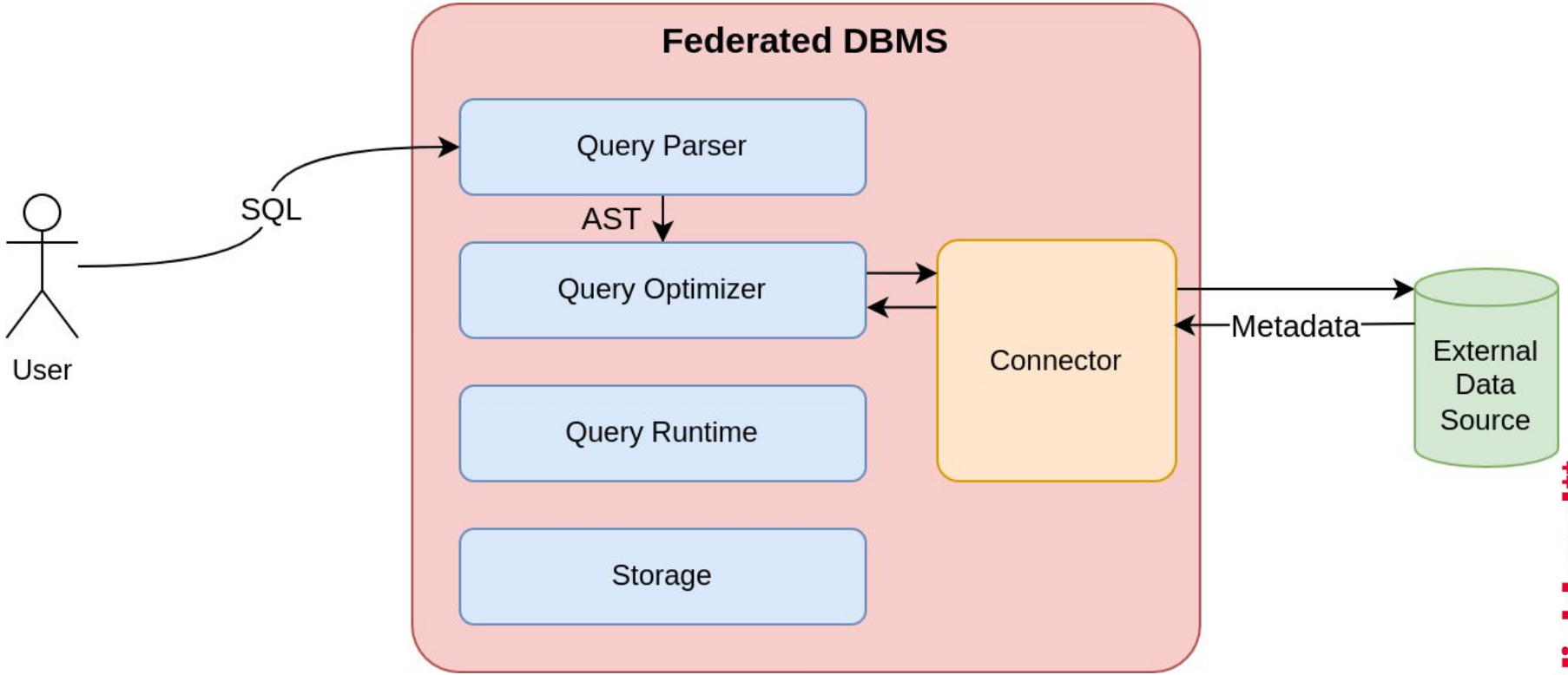
Обработка запроса



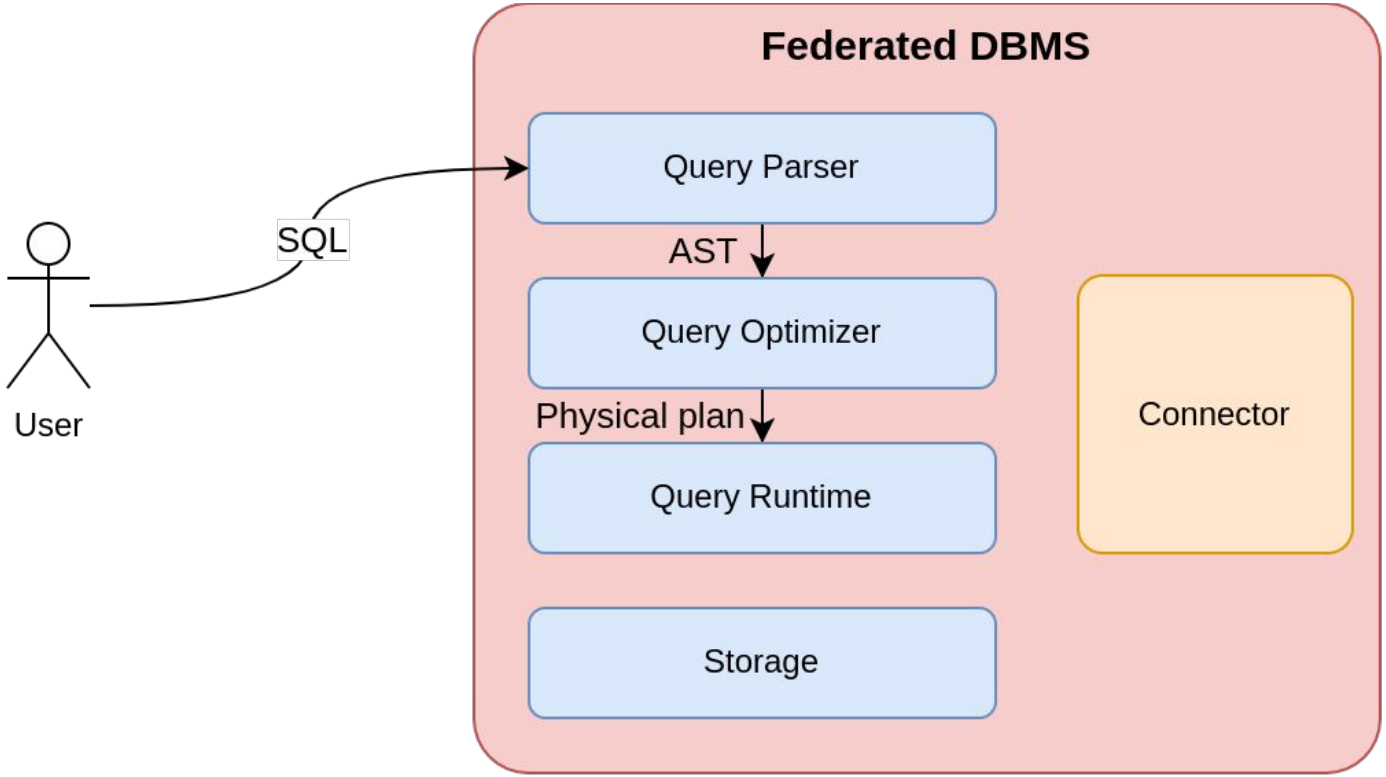
Парсинг запроса



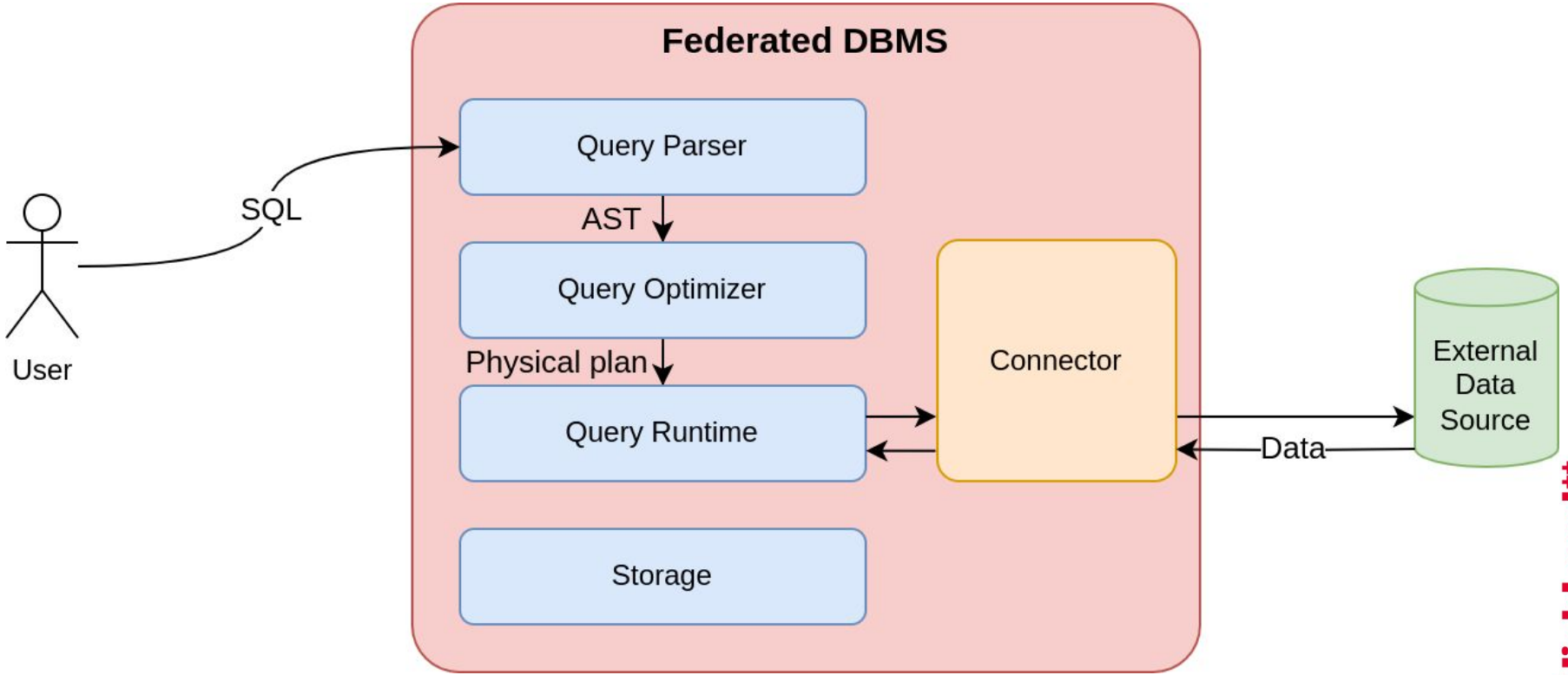
Извлечение метаданных



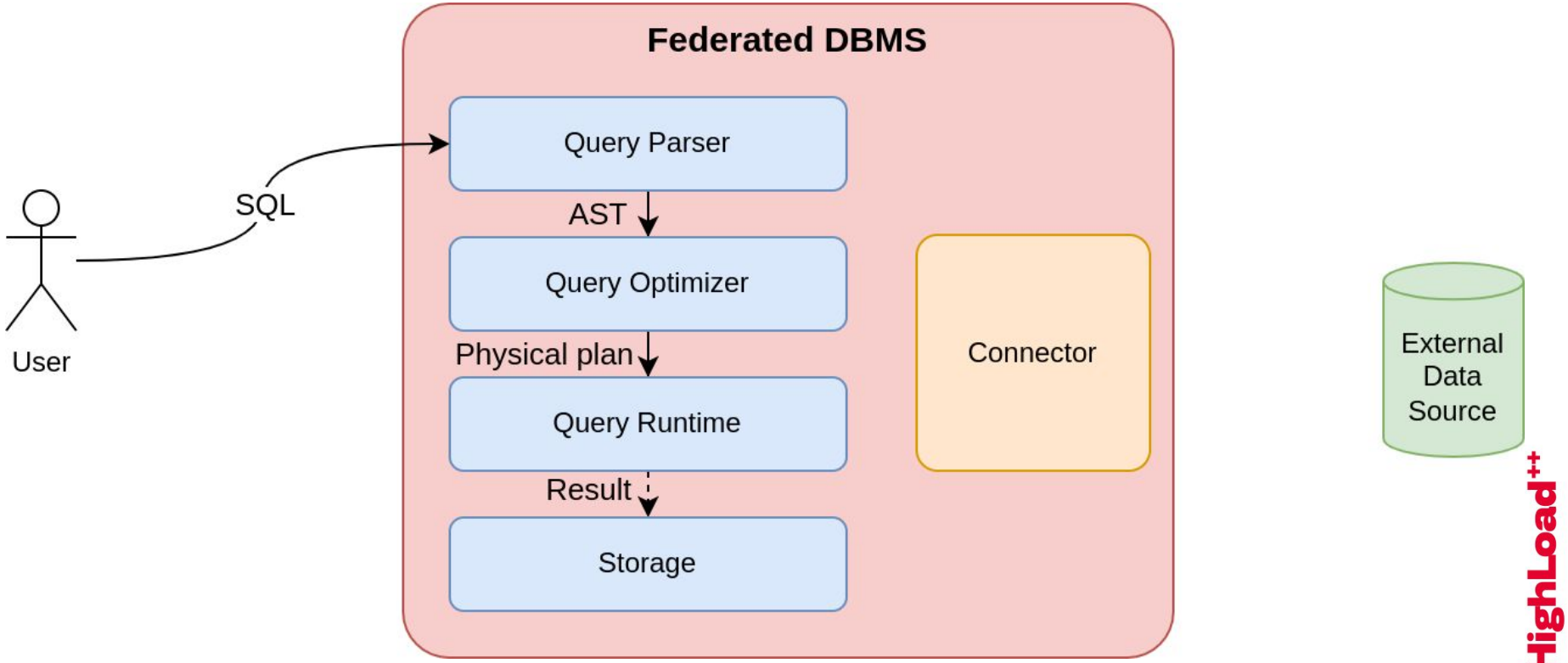
Физический план выполнения



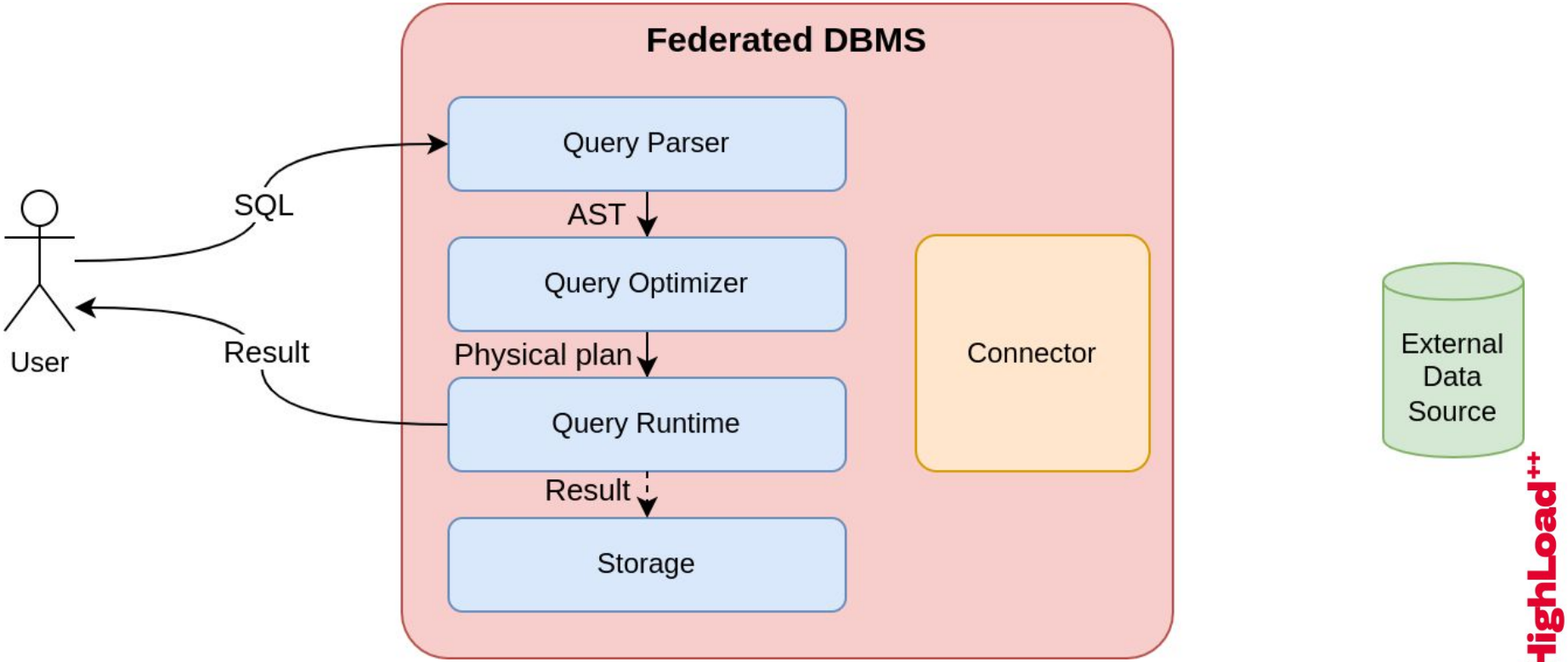
Извлечение данных



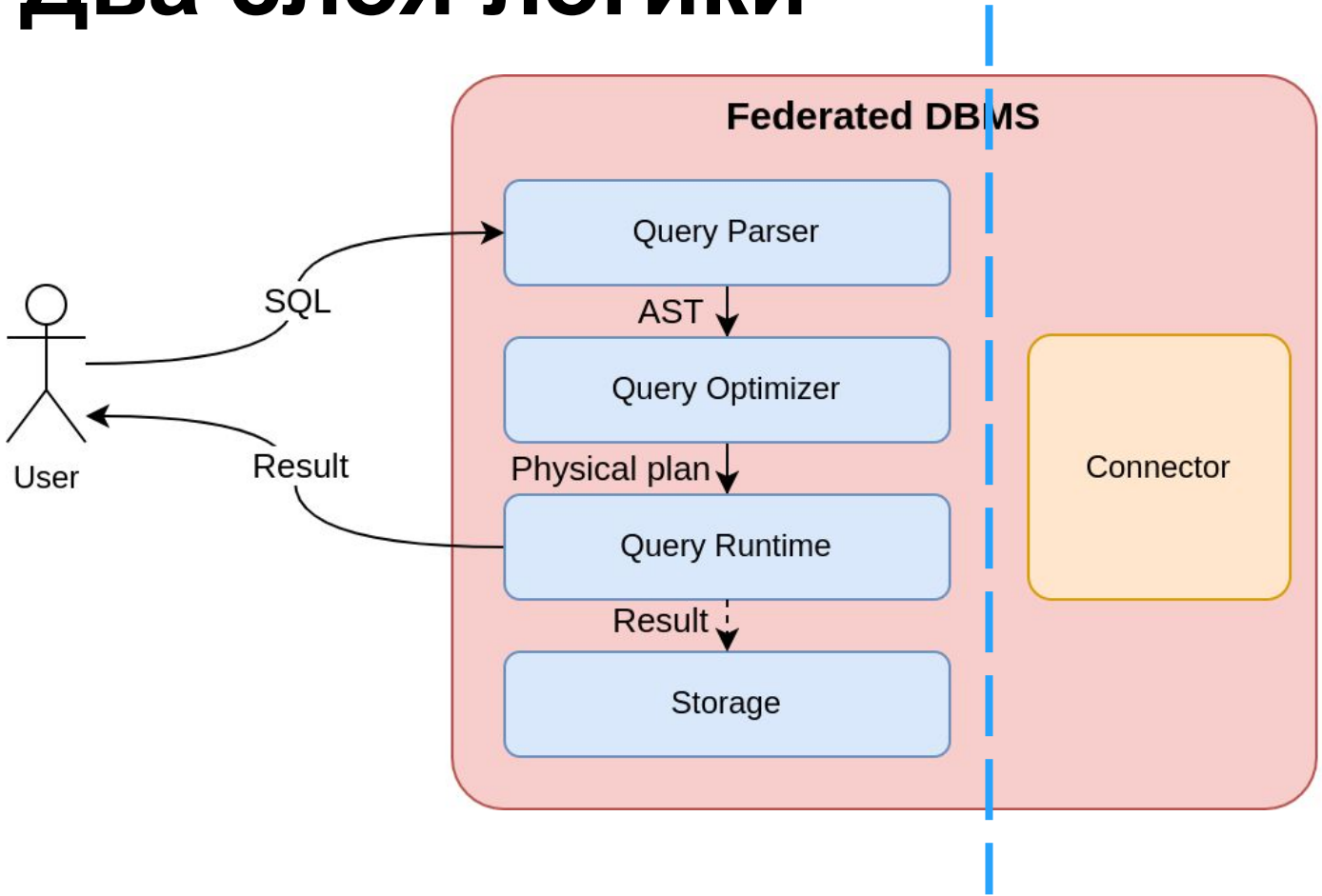
Сохранение результата



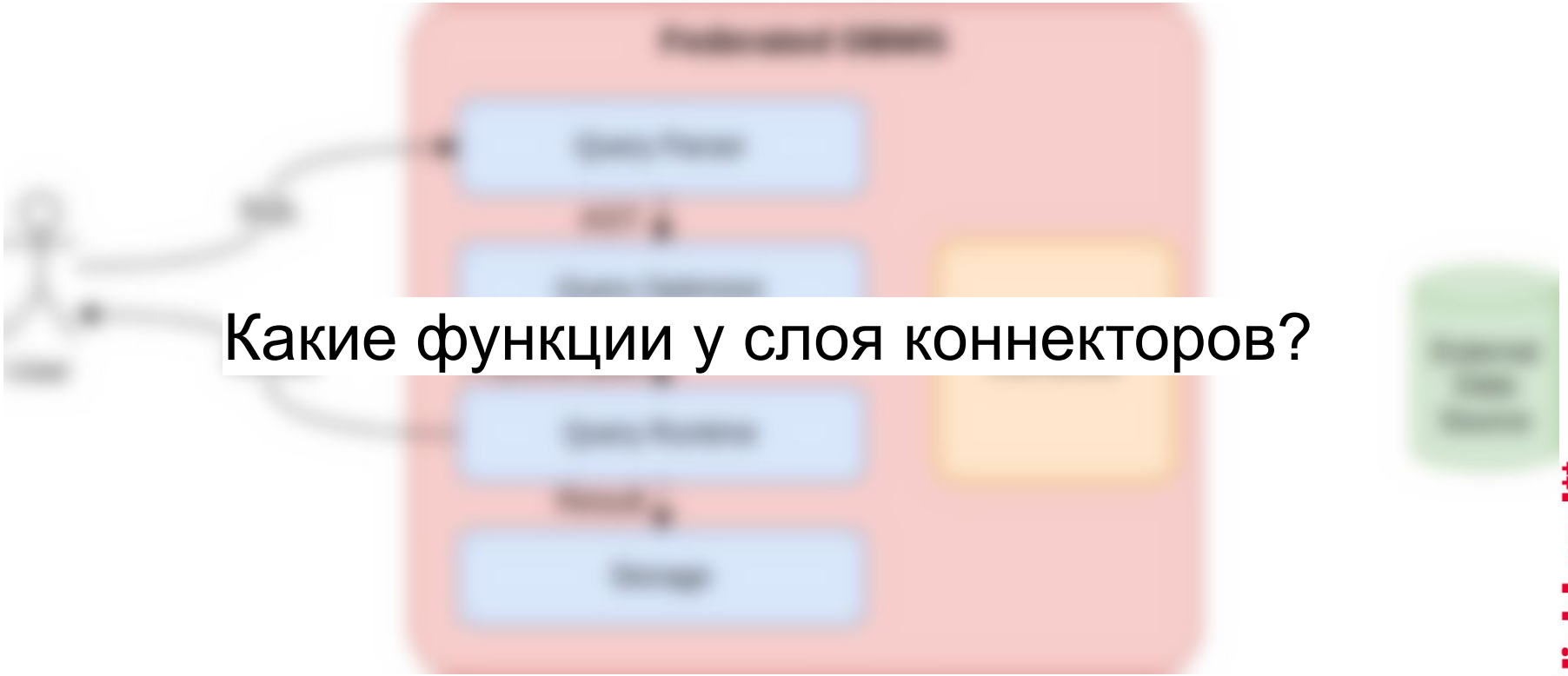
Ответ пользователю



Два слоя логики

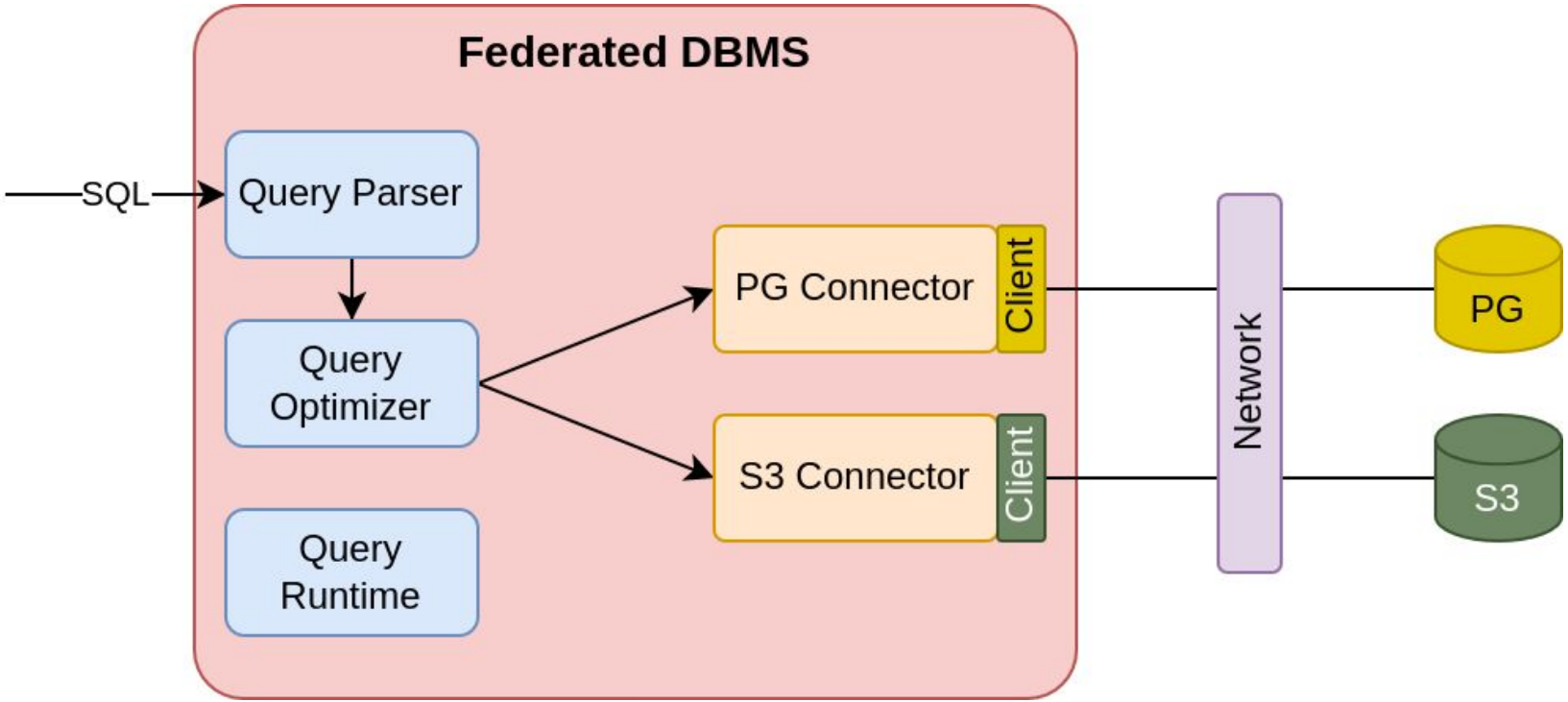


Два слоя логики

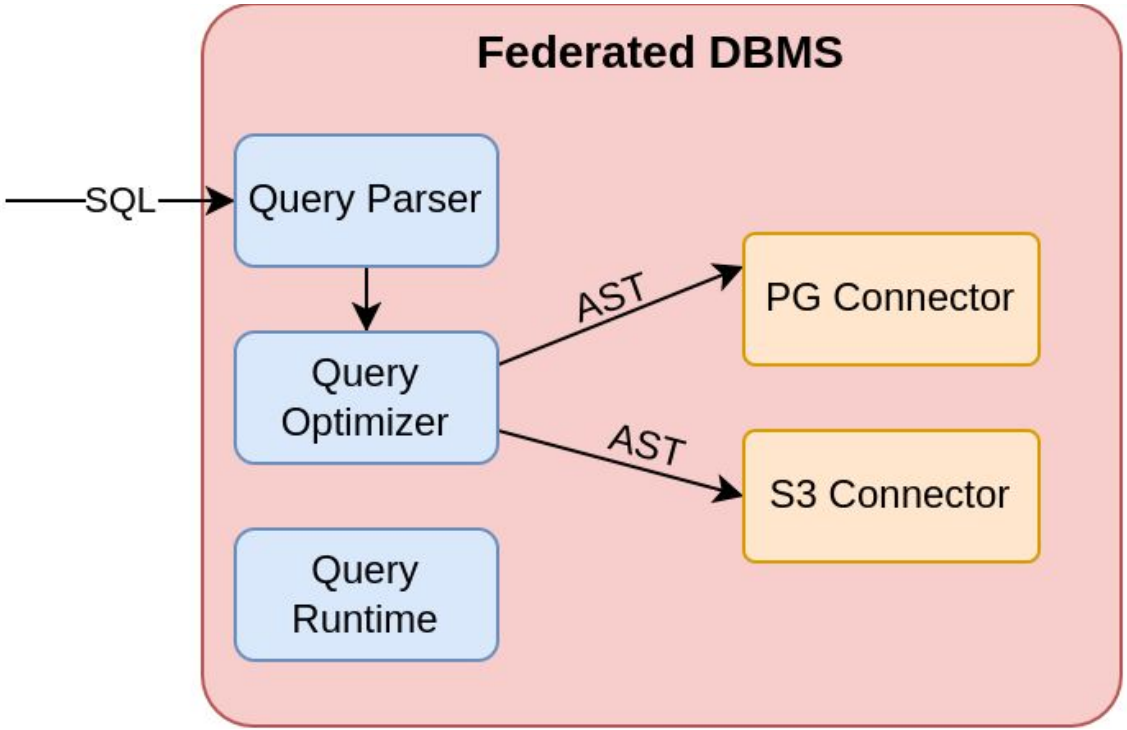


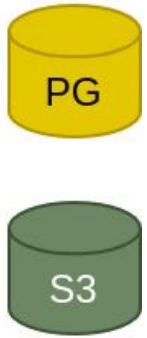
Какие функции у слоя коннекторов?

Сеть, шифрование, ретраи



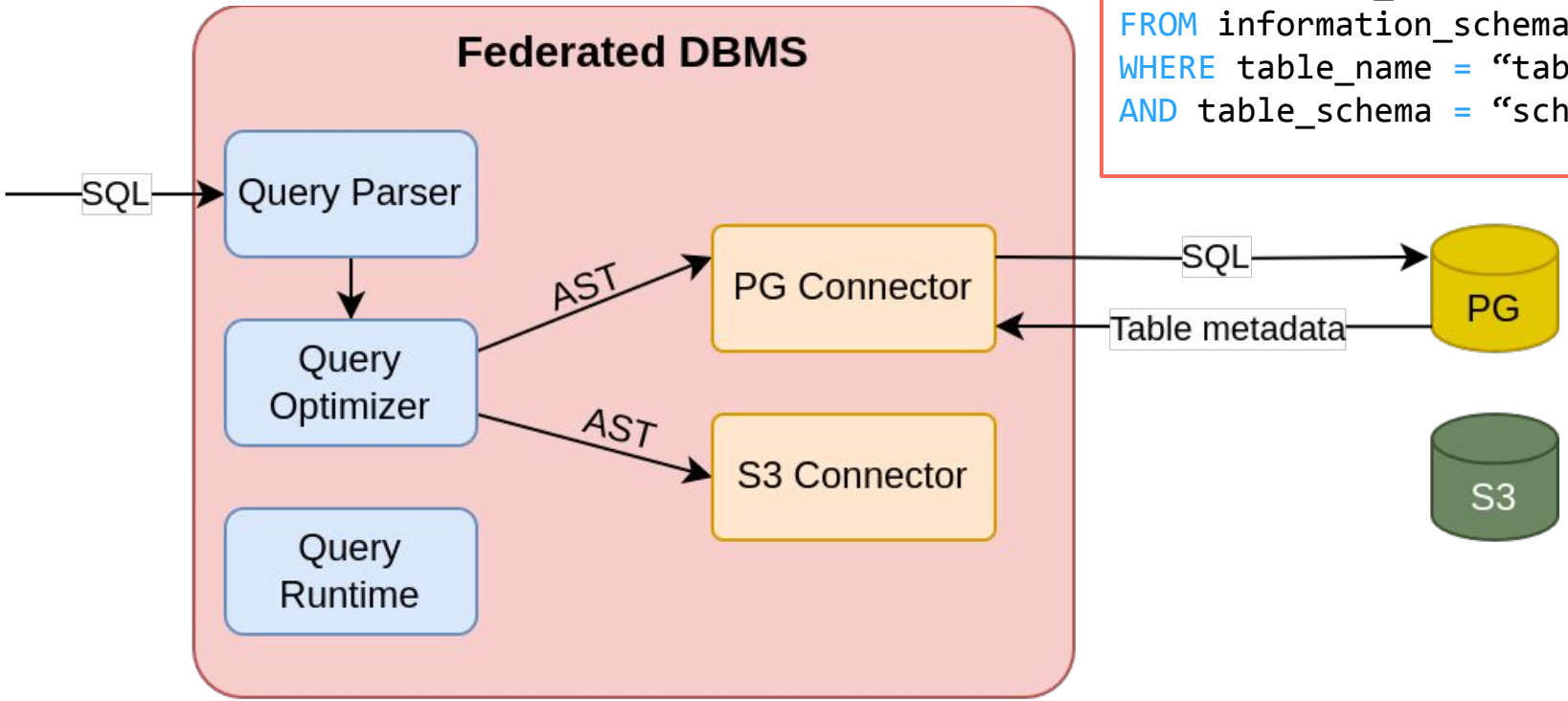
Трансляция запросов и схем



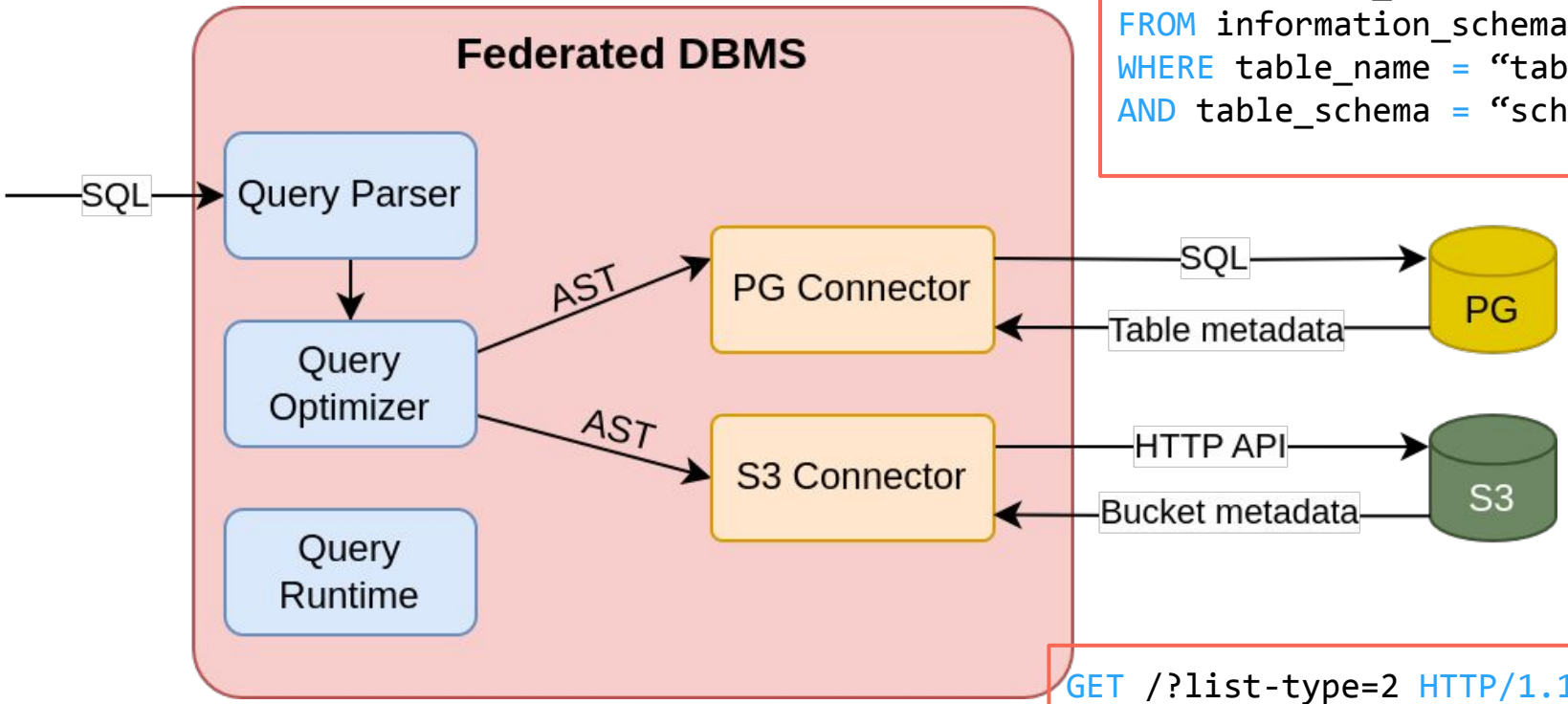


Трансляция запросов и схем


```
SELECT column_name, data_type
FROM information_schema.columns
WHERE table_name = “table”
AND table_schema = “schema”
```



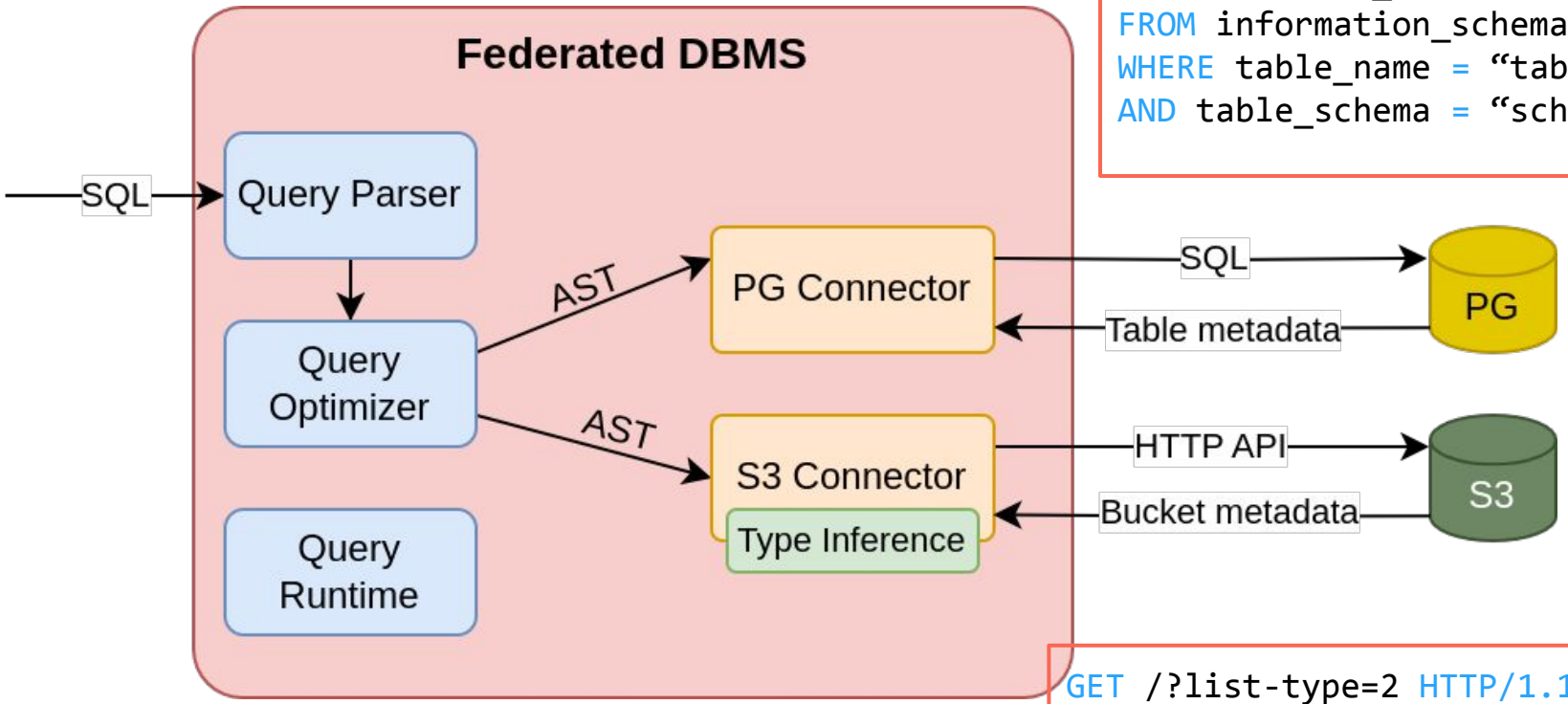
Трансляция запросов и схем




```
SELECT column_name, data_type
FROM information_schema.columns
WHERE table_name = “table”
AND table_schema = “schema”
```

```
GET /?list-type=2 HTTP/1.1
Host:
bucket.storage.yandexcloud.net
```

Трансляция запросов и схем



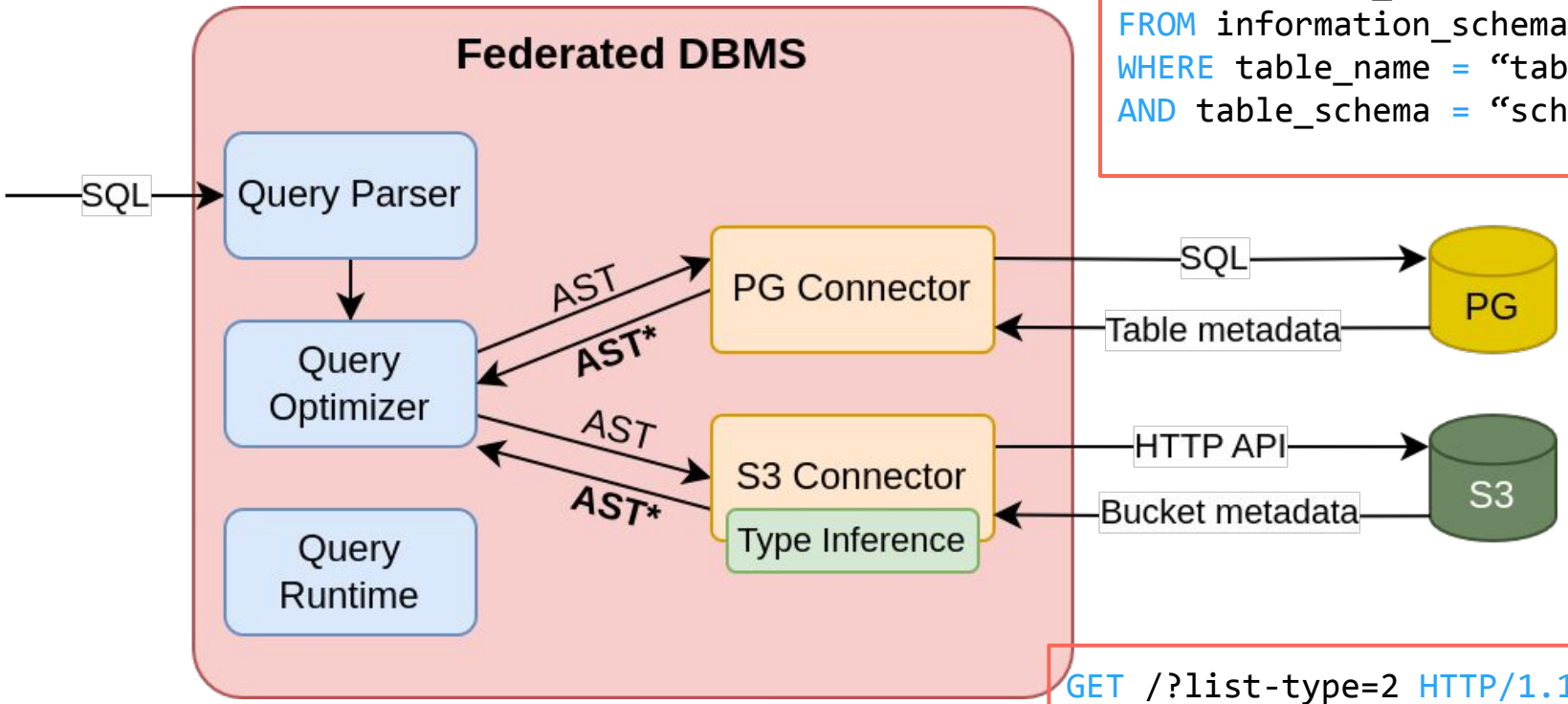

```
SELECT column_name, data_type
FROM information_schema.columns
WHERE table_name = “table”
AND table_schema = “schema”
```

```
GET /?list-type=2 HTTP/1.1
Host:
bucket.storage.yandexcloud.net
```

HighLoad++ 2024

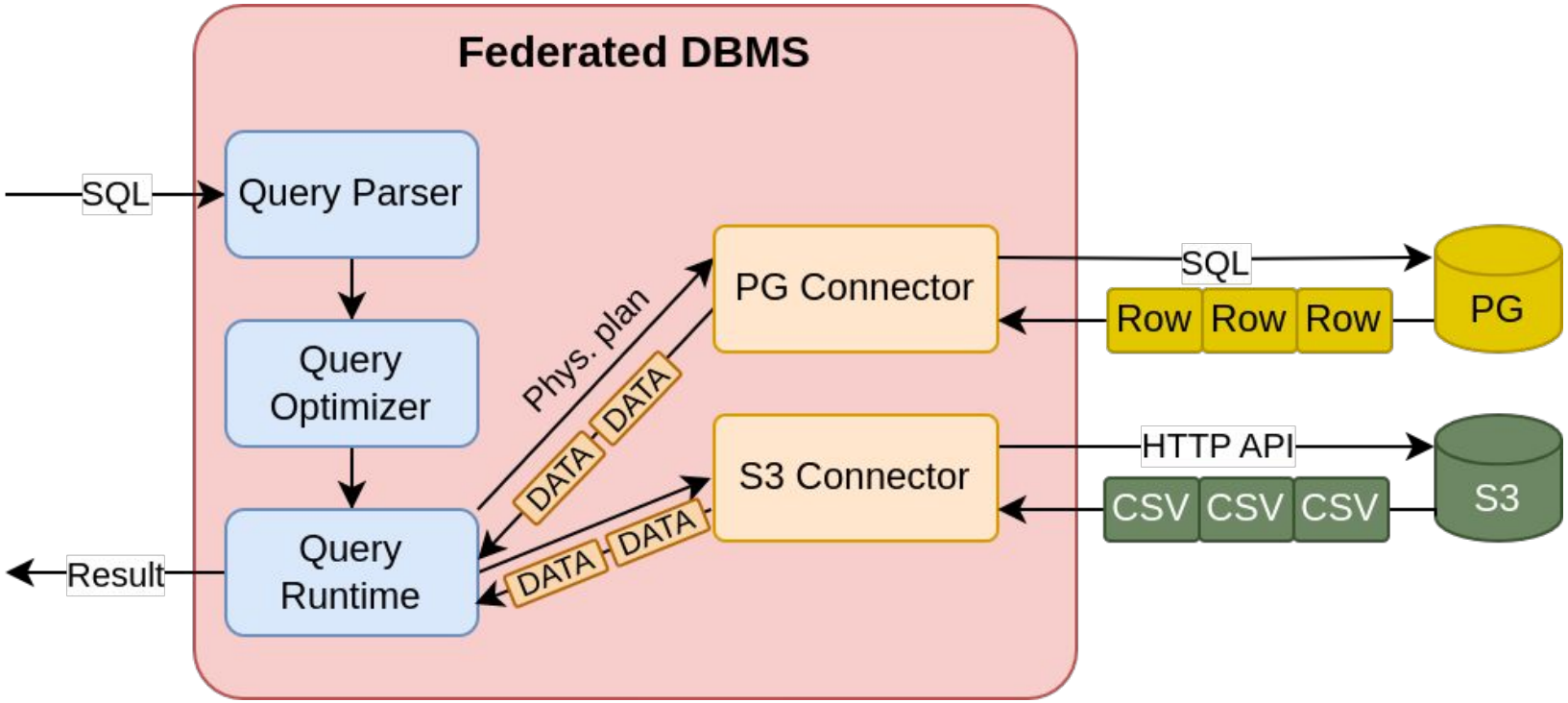
Трансляция запросов и схем


```
SELECT column_name, data_type
FROM information_schema.columns
WHERE table_name = “table”
AND table_schema = “schema”
```




```
GET /?list-type=2 HTTP/1.1
Host:
bucket.storage.yandexcloud.net
```

Унификация данных

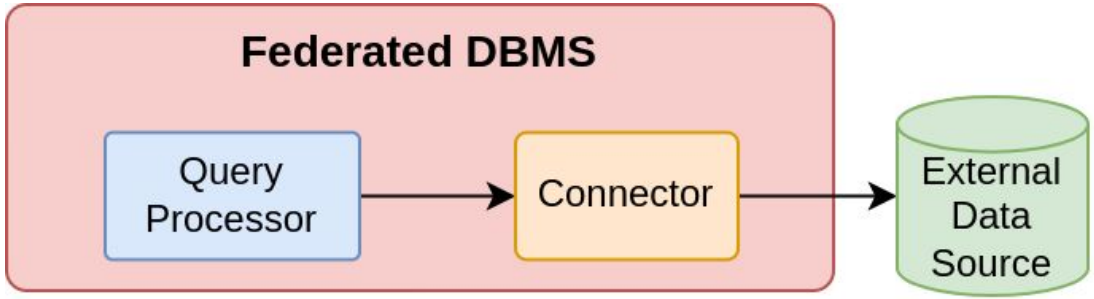


Как устроены коннекторы к внешним источникам

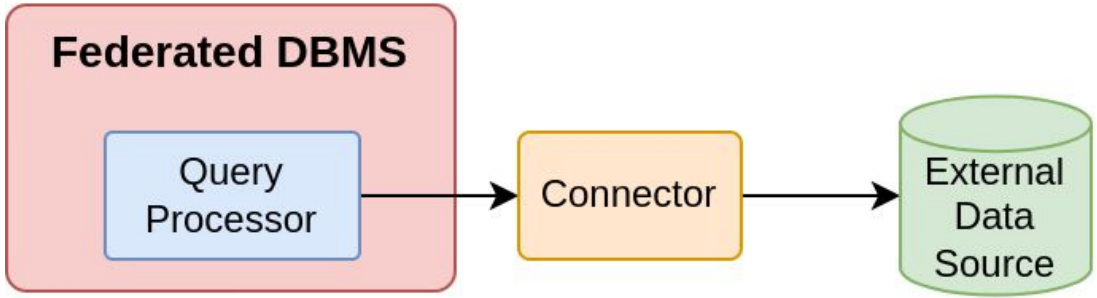
HL 2024 HighLoad++

Встроенный или внешний?

Встроенный коннектор



Внешний коннектор



Встроенный или внешний?

Характеристика	**Встроенный коннектор**	**Внешний коннектор**
Быстродействие	➕	➖

Встроенный или внешний?

Характеристика	**Встроенный коннектор**	**Внешний коннектор**
Быстродействие	➕	➖
Удобство развёртывания	➕	➖

Встроенный или внешний?

Характеристика	**Встроенный коннектор**	**Внешний коннектор**
Быстродействие	➕	➖
Удобство развёртывания	➕	➖
Масштабируемость	➖	➕

Встроенный или внешний?

Характеристика	**Встроенный коннектор**	**Внешний коннектор**
Быстродействие	+	−
Удобство развёртывания	+	−
Масштабируемость	−	+
Выбор ЯП и скорость разработки	−	+

Встроенный или внешний?

Характеристика	**Встроенный коннектор**	**Внешний коннектор**
Стабильность СУБД (C/C++)		

Встроенный или внешний?

Характеристика	**Встроенный коннектор**	**Внешний коннектор**
Стабильность СУБД (C/C++)	➖	➕
Скорость сборки СУБД (C/C++)	➖	➕
Размер бинарника (C/C++)	➖	➕

Presto, Trino, AWS Athena



Presto, Trino, AWS Athena



Presto/Trino — встроенные:

- коннектор — плагин к движку
- Service Provider Interface

Presto, Trino, AWS Athena



Presto/Trino — встроенные:

- коннектор — плагин к движку
- Service Provider Interface

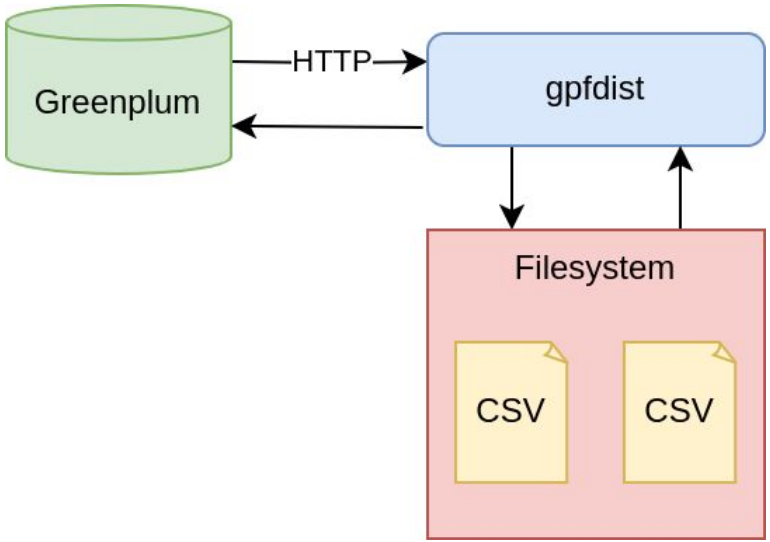
AWS Athena — внешние:

- большинство коннекторов — serverless-приложения
- коннектор к S3 — ?

Greenplum

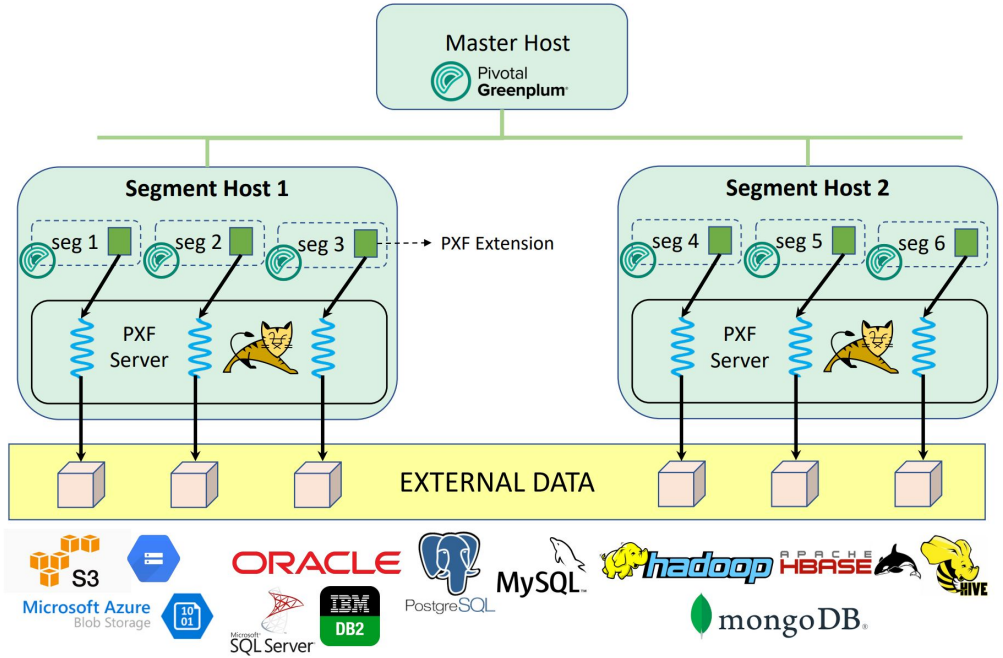
Greenplum

- gpfdist
  (HTTP over FS)



Greenplum

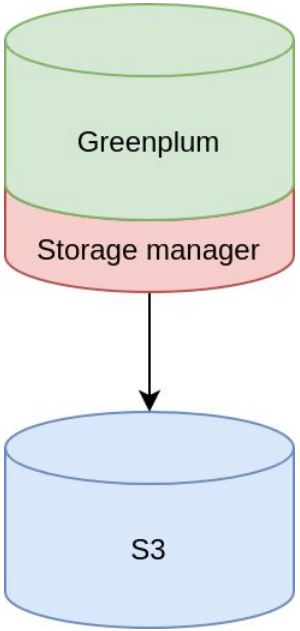
- gpfdist
  (HTTP over FS)
- PXF
  (Platform Extension
  Framework)



Источник: [Platform Extension Framework (PXF): Enabling Parallel Query Processing Over Heterogeneous Data Sources in Greenplum, 2020]

Greenplum

- gpfdist
  (HTTP over FS)
- PXF
  (Platform Extension Framework)
- yezzey
  (Pluggable SMGR -> S3)



Clickhouse

Встроенные коннекторы:

- Integration Table Engine
(13 источников)
- Database Engine
(3 источника)

Clickhouse

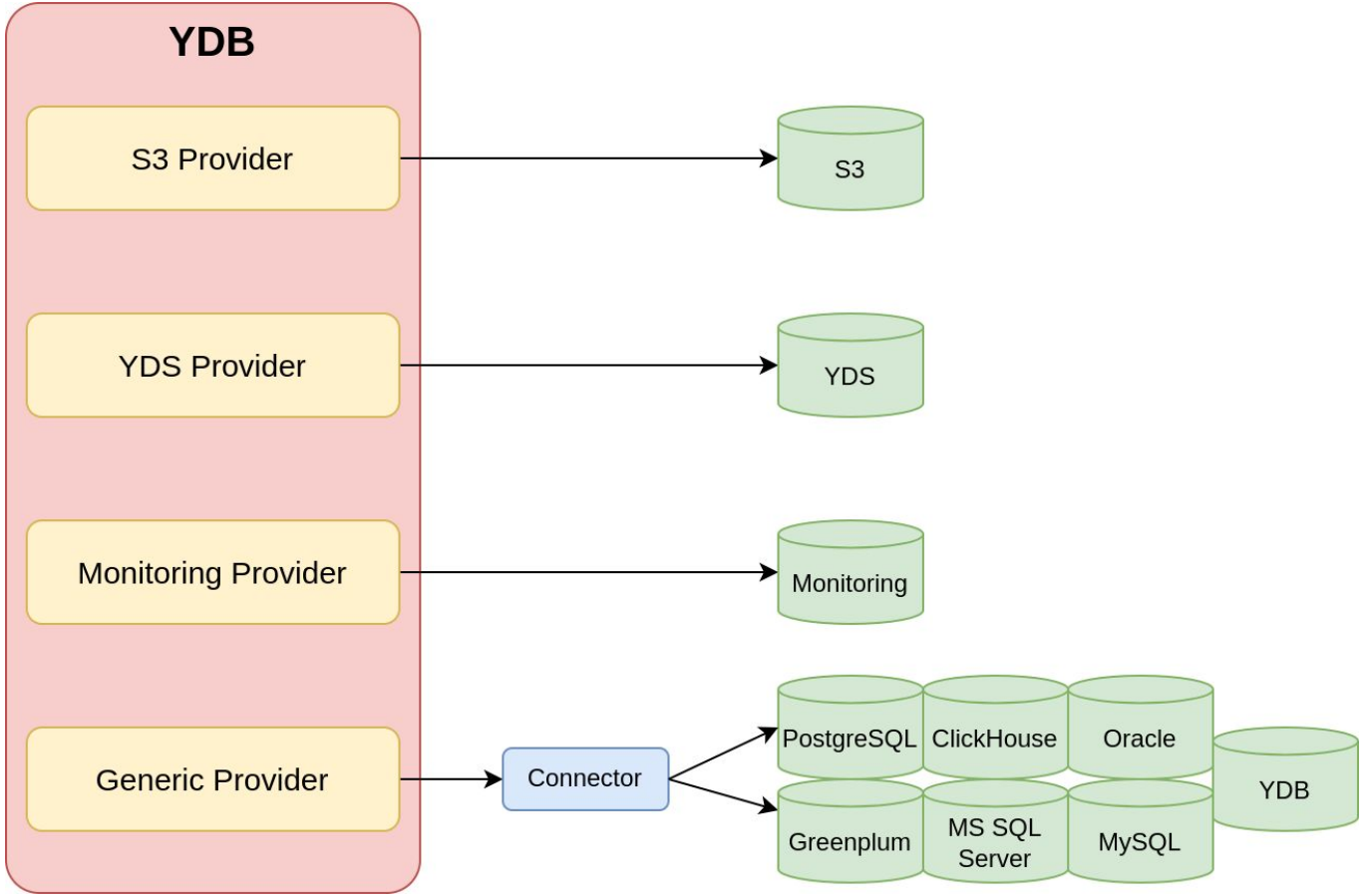
Встроенные коннекторы:

- Integration Table Engine (13 источников)
- Database Engine (3 источника)

Внешние коннекторы:

- ODBC Bridge
- JDBC Bridge (deprecated)

YDB



Tradeoff #1: архитектура

Система	Встроенные коннекторы	Внешние коннекторы
 Presto	✅	❌
 Trino	✅	❌
 AWS Athena	🤔	✅
 Greenplum	🤔	✅
 ClickHouse	✅	✅
 YDB	✅	✅

Tradeoff #1: архитектура

Система	Встроенные коннекторы	Внешние коннекторы
 Presto	✅	❌
 Trino	✅	❌
 AWS Athena	🤔	✅
 Greenplum	🤔	✅
 ClickHouse	✅	✅
 YDB	✅	✅

Как читать как можно меньше данных

HL 2024 HighLoad++

Запросы с предикатами


```
SELECT * FROM ext_source.table WHERE id = 1
```

Предикат — логическое выражение, которое используется для описания условия и применяется к строкам в таблице.

Запросы с предикатами


```
SELECT * FROM ext_source.table WHERE id = 1
```

Предикат — логическое выражение, которое используется для описания условия и применяется к строкам в таблице.

Где выполнять фильтрацию?

- На стороне федеративной системы
- На стороне внешнего источника **(пушдаун предикатов)**

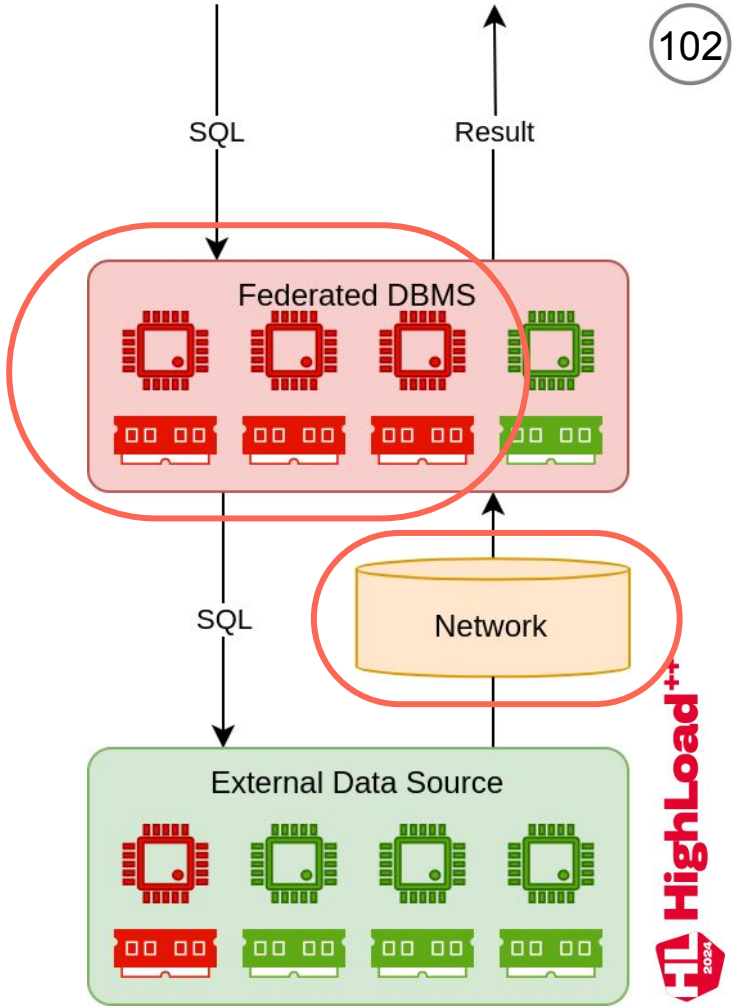
Пушдаун предиката ❌

Федеративный запрос:

```
SELECT * FROM ext_source.table
WHERE id = 1
```

Запрос к источнику:

```
SELECT * FROM table
```



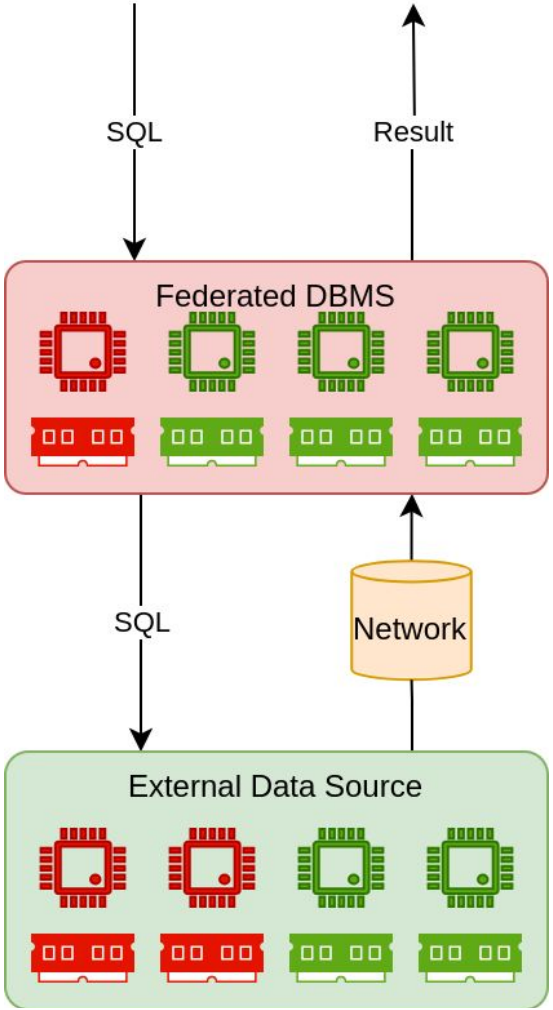
Пушдаун предиката ✅

Федеративный запрос:

```
SELECT * FROM ext_source.table
WHERE id = 1
```

Запрос к источнику:

```
SELECT * FROM table
WHERE id = 1
```



Виды пушдаунов

Пушдаун фильтров:

- Comparison operators:
  `< <= = != >= >`
  `BETWEEN IN IS NULL`
- Logical operators:
  `AND OR NOT`
- Math operators:
  `+ - / * % | & ^`
- Pattern matching: `LIKE`

Виды пушдаунов

Пушдаун фильтров:

- Comparison operators:
  `< <= = != >= >`
  `BETWEEN IN IS NULL`
- Logical operators:
  `AND OR NOT`
- Math operators:
  `+ - / * % | & ^`
- Pattern matching: `LIKE`

Прочее:

- Column projection:
  `SELECT a, b FROM table`
- `JOIN ... ON a = b`
- `ORDER BY a [DESC|ASC]`
- `LIMIT ... OFFSET ...`
- Aggregates: `COUNT SUM AVG`
- Subqueries

Пушдаун для SQL-источников

Система	Filters	Column proj.	JOIN	ORDER BY	LIMIT	Aggregates	Subqueries
Presto							
Trino							
AWS Athena							
Greenplum (PXF)							
ClickHouse (Table engines)							
ClickHouse (JDBC, ODBC)							
YDB							

Реализовано для некоторых источников	Упоминания не найдены

Пушдаун для SQL-источников

Система	Filters	Column proj.	JOIN	ORDER BY	LIMIT	Aggregates	Subqueries
Presto							
Trino							
AWS Athena							
Greenplum (PXF)							
ClickHouse (Table engines)							
ClickHouse (JDBC, ODBC)							
YDB							

Почему так происходит?

Реализовано для некоторых источников	Упоминания не найдены

AST простого запроса

YQL:

```
SELECT 1
```

AST:

```
(
(let $1 (Configure! world (DataSource '"config") '"DqEngine"
'"auto"))
(let $2 (DataSink 'result))
(let $3 '('('type) '('autoref) '('columns '('"column0"))))
(let $4 (ResFill! $1 $2 (Key) (AsList (AsStruct '('"column0"
(Int32 '"1")))) $3 '"dq"))
(return (Commit! $4 $2))
)
```

AST запроса с пушдауном

YQL:

```
SELECT col1 FROM mysql.table WHERE id = 1
```

AST:

```
(
(let $1 '"/dc1/org12345/folder12345/mysql")
(let $2 (DataSource '"generic" $1))
(let $3 '"cluster:default_/dc1/org12345/folder12345/mysql")
(let $4 (Int32 '1))
(let $5 (Bool 'false))
(let $6 (GenSourceSettings $1 '"table" (SecureParam $3) '('"col1" '"id") (lambda '($13) (Coalesce (== (Member $13 '"id") $4) $5))))
(let $7 '('"col1" (OptionalType (DataType 'Utf8))))
(let $8 (Uint64 '"1000001"))
(let $9 (DqPhyStage '((DqSource $2 $6)) (lambda '($14) (FromFlow (Take (Map (Filter (NarrowMap (WideFromBlocks
(DqSourceWideBlockWrap $14 $2 (StructType $7 '('"id" (OptionalType (DataType 'Int32)))))) (lambda '($15 $16) (AsStruct '('"col1"
$15) '('"id" $16)))) (lambda '($17) (Coalesce (== (Member $17 '"id") $4) $5))) (lambda '($18) (AsStruct '('"col1" (Member $18
'"col1"))))) $8))) '('('"_logical_id" '595) '('"_id" '"bdc3e1c8-16dd2698-463253f8-6b9b3fe2"))))
(let $10 (DqCnUnionAll (TDqOutput $9 '"0")))
(let $11 (DqPhyStage '($10) (lambda '($19) (FromFlow (Take (ToFlow $19) $8))) '('('"_logical_id" '608) '('"_id"
'"137c11d-a4fe308d-81e00b79-b51ed589"))))
(let $12 (DqCnResult (TDqOutput $11 '"0") '('"col1")))
(return (KqpPhysicalQuery '((KqpPhysicalTx '($9 $11) '($12) '() '('('"type" '"generic")))) '((KqpTxResultBinding (ListType
(StructType $7)) '"0" '"0")) '('('"type" '"script"))))
)
```

Athena: API коннектора без AST


```
public PreparedStatement prepareStatementWithSql(
     final Connection jdbcConnection, /* сетевое соединение */
     final String schema,             /* имя схемы (неймспейса) таблиц */
     final String table,              /* имя таблицы */
     final Schema tableSchema,        /* имена колонок */
     final Constraints constraints,   /* элементы запроса для pushdown */
     final Split split)               /* параметр партиционирования */
     {
          StringBuilder sql = new StringBuilder();
          sql.append("SELECT ");

          /* Добавляем column projection */
          sql.append(columnNamesFromSchema(tableSchema));

          /* Добавляем FROM */
          sql.append(" FROM ");
          sql.append(getFromClauseWithSplit(catalog, schema, table, split));
```

Athena: API коннектора без AST


```
public PreparedStatement prepareStatementWithSql(
     final Connection jdbcConnection, /* сетевое соединение */
     final String schema,             /* имя схемы (неймспейса) таблиц */
     final String table,              /* имя таблицы */
     final Schema tableSchema,        /* имена колонок */
     final Constraints constraints,   /* элементы запроса для pushdown */
     final Split split)               /* параметр партиционирования */
     {
          StringBuilder sql = new StringBuilder();
          sql.append("SELECT ");

          /* Добавляем column projection */
          sql.append(columnNamesFromSchema(tableSchema));

          /* Добавляем FROM */
          sql.append(" FROM ");
          sql.append(getFromClauseWithSplit(catalog, schema, table, split));
```

Athena: API коннектора без AST


```
public PreparedStatement prepareStatementWithSql(
     final Connection jdbcConnection, /* сетевое соединение */
     final String schema,             /* имя схемы (неймспейса) таблиц */
     final String table,              /* имя таблицы */
     final Schema tableSchema,        /* имена колонок */
     final Constraints constraints,   /* элементы запроса для pushdown */
     final Split split)               /* параметр партиционирования */
     {
          StringBuilder sql = new StringBuilder();
          sql.append("SELECT ");

          /* Добавляем column projection */
          sql.append(columnNamesFromSchema(tableSchema));

          /* Добавляем FROM */
          sql.append(" FROM ");
          sql.append(getFromClauseWithSplit(catalog, schema, table, split));
```

Athena: API коннектора без AST


```
public PreparedStatement prepareStatementWithSql(
     final Connection jdbcConnection, /* сетевое соединение */
     final String schema,             /* имя схемы (неймспейса) таблиц */
     final String table,              /* имя таблицы */
     final Schema tableSchema,        /* имена колонок */
     final Constraints constraints,   /* элементы запроса для pushdown */
     final Split split)               /* параметр партиционирования */
     {
          StringBuilder sql = new StringBuilder();
          sql.append("SELECT ");

          /* Добавляем column projection */
          sql.append(columnNamesFromSchema(tableSchema));

          /* Добавляем FROM */
          sql.append(" FROM ");
          sql.append(getFromClauseWithSplit(catalog, schema, table, split));
```

Athena: API коннектора без AST


```
public PreparedStatement prepareStatementWithSql(
     final Connection jdbcConnection, /* сетевое соединение */
     final String schema,             /* имя схемы (неймспейса) таблиц */
     final String table,              /* имя таблицы */
     final Schema tableSchema,        /* имена колонок */
     final Constraints constraints,   /* элементы запроса для pushdown */
     final Split split)               /* параметр партиционирования */
     {
          StringBuilder sql = new StringBuilder();
          sql.append("SELECT ");

          /* Добавляем column projection */
          sql.append(columnNamesFromSchema(tableSchema));

          /* Добавляем FROM */
          sql.append(" FROM ");
          sql.append(getFromClauseWithSplit(catalog, schema, table, split));
```

Athena: API коннектора без AST


```
/* Добавляем WHERE */
List<String> clauses = toConjuncts(tableSchema.getFields(), constraints,
     split.getProperties());
clauses.addAll(getPartitionWhereClauses(split));
if (!clauses.isEmpty()) {
     sql.append(" WHERE ").append(Joiner.on(" AND ").join(clauses));
}
/* Добавляем ORDER BY */
String orderByClause = extractOrderByClause(constraints);
if (!Strings.isNullOrEmpty(orderByClause)) {
     sql.append(" ").append(orderByClause);
}
/* Добавляем LIMIT/OFFSET */
if (constraints.getLimit() > 0) {
     sql.append(appendLimitOffset(split, constraints));
}

LOGGER.debug("Generated SQL : {}", sql.toString());
```

Tradeoff #2: интерфейс

Federated DBMS / Engine	Интерфейс коннектора	
	Гибкий (с AST)	Ограниченный (без AST)
Presto	❌	✅
Trino	❌	✅
AWS Athena	❌	✅
Greenplum	🤔	✅
ClickHouse	✅ (Table Engines)	✅ (ODBC)
YDB	✅ (S3, YDS)	✅ (RDBMS)

Итоги

Не успели обсудить

- Массивно-параллельное чтение из внешних источников
- Скалярные vs блочные вычисления
- Алгоритмы JOIN
- Спиллинг данных

Критерии выбора

- Спектр поддерживаемых источников
- Полнота системы типов
- Архитектура и интерфейс коннекторов
- Pushdown запросов

On-prem-решения

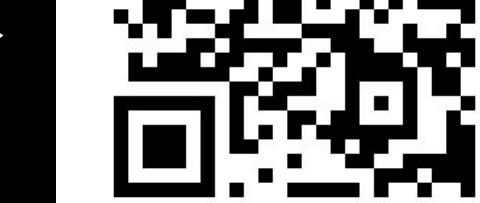
- Trino
- ClickHouse
- YDB

YDB Federated Query Connector

Облачные решения

- **Yandex Query** (бессерверная аналитика)
- **Yandex Data Transfer** (перенос данных между хранилищами)
- **Yandex Managed Service For Trino** (кластер Trino в облаке)

Голосуйте за мой доклад



Виталий Исаев

@vent_2

HL 2024 HighLoad++