VERTICA

How to bring advanced analytics to hybrid data storage with Vertica

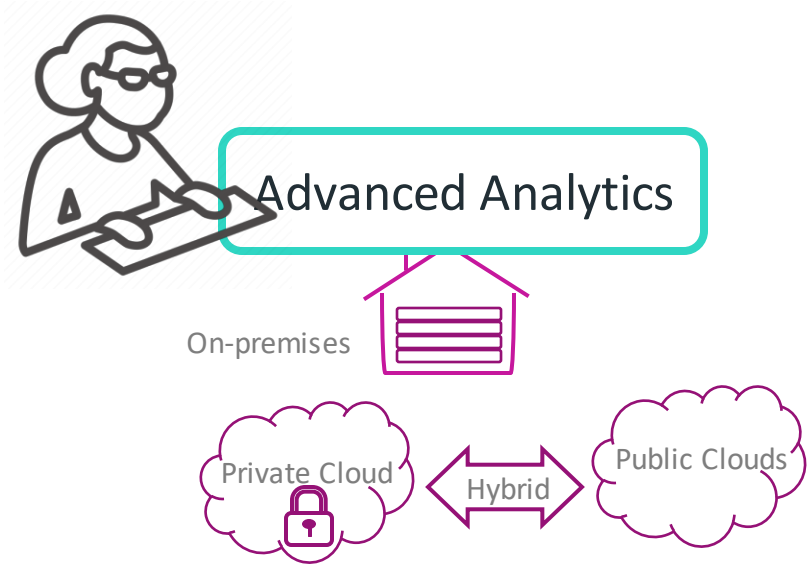
Marco Gessner
marco.Gessner@vertica.com

Gianluigi Vigano'
gianluigi.vigano@vertica.com

Maurizio Felici
maurizio.felici@vertica.com

MICRO FOCUS

Advanced Analytics in the hybrid/distributed Age

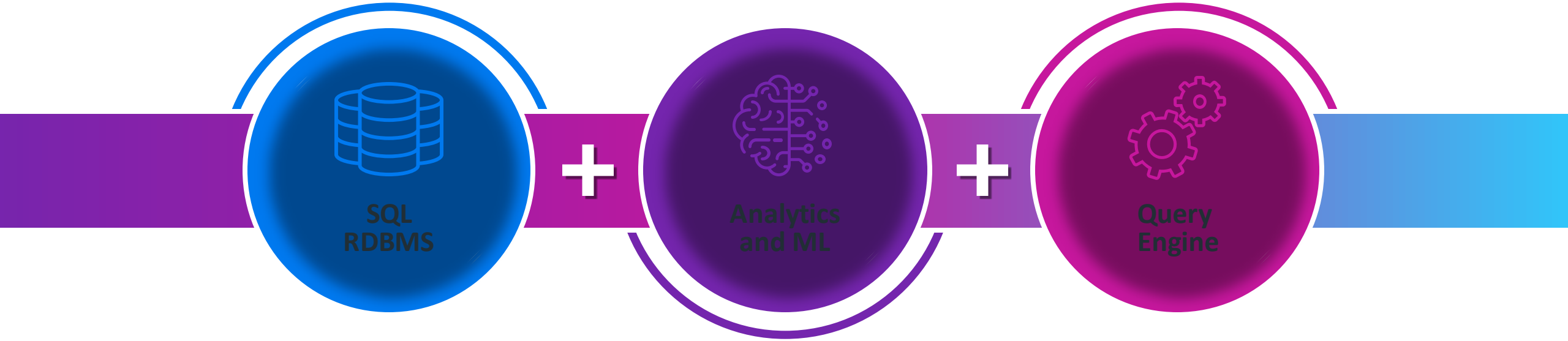


VERTICA

Vertica, a unified Advanced Analytics engine

What is Vertica?

Vertica is the first database built completely **columnar**.
It is an **MPP** query engine with advanced analytics and machine learning.

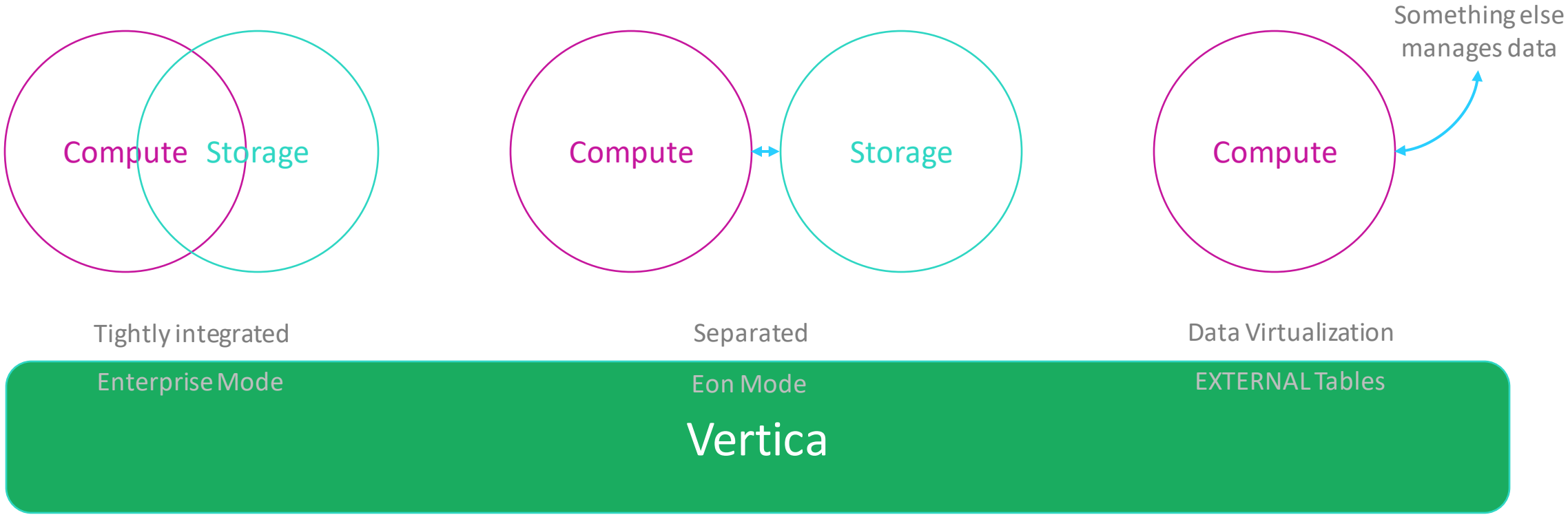


Load and store data in a data warehouse designed for **blazing fast analytics**

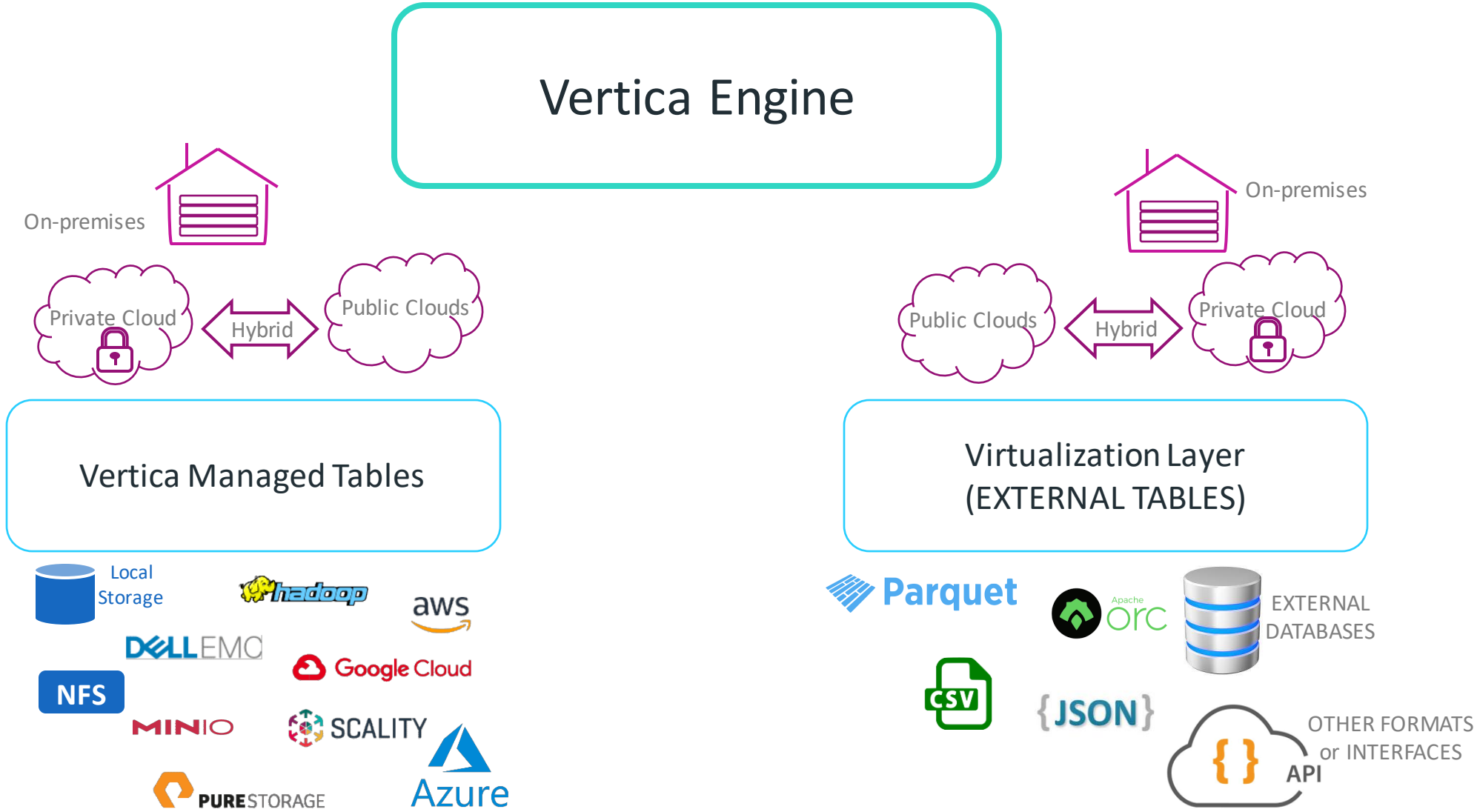
Create, **train,** and **deploy** advanced analytics and **machine learning** models at massive scale

Ask complex analytical questions and get **fast answers**, regardless of where the data resides

Compute, Storage and Big Data

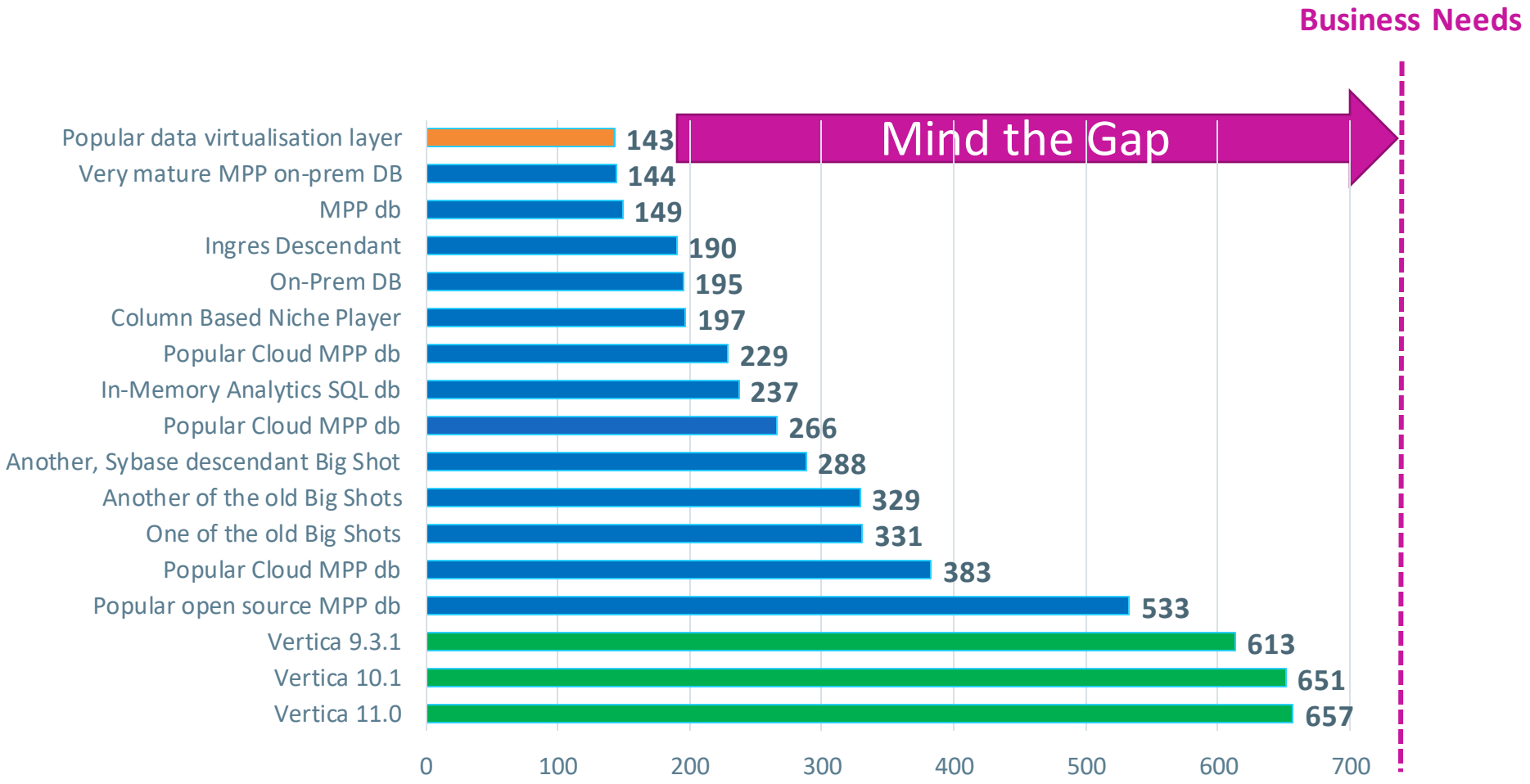


Vertica Engine



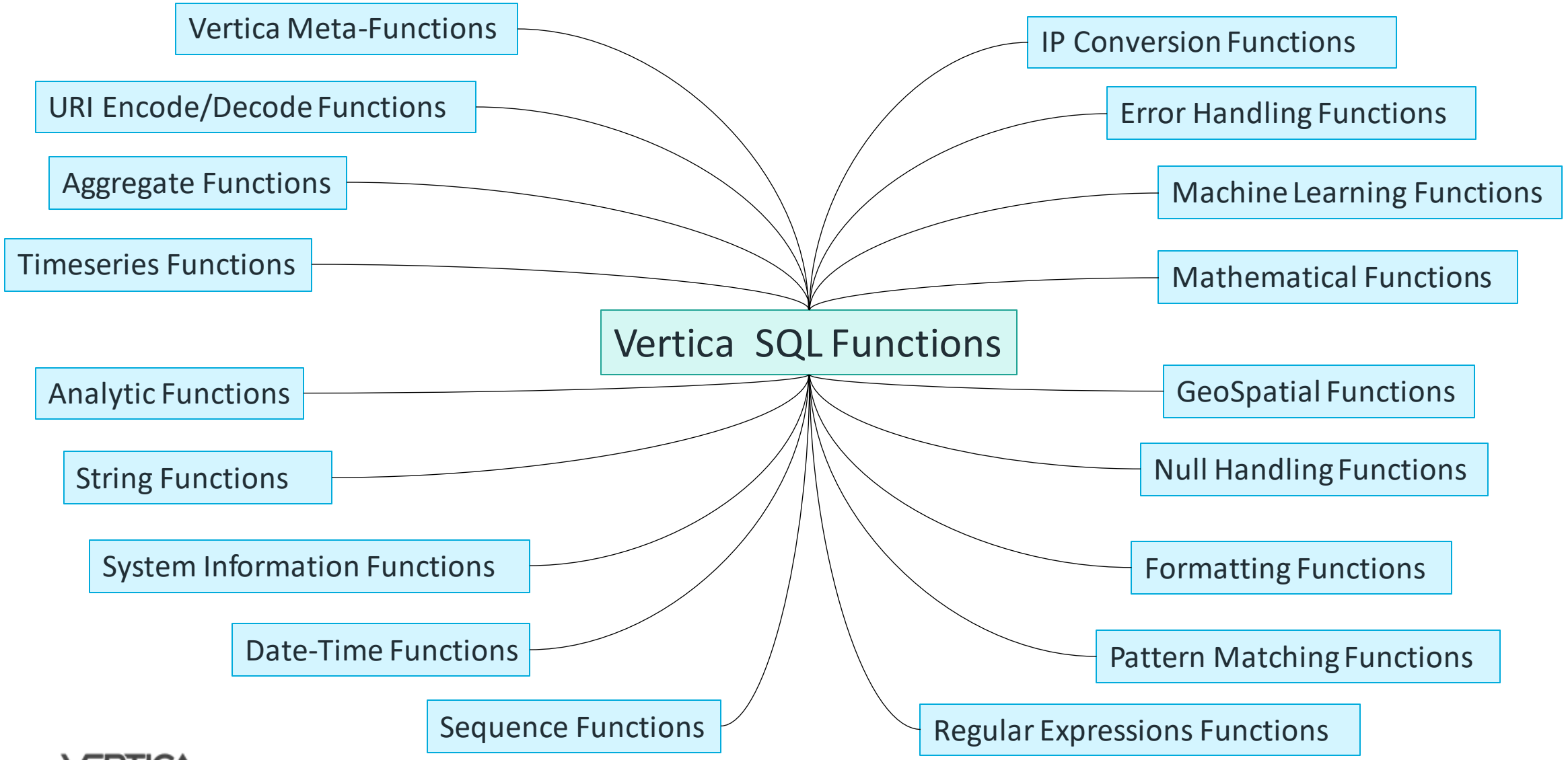
VERTICA

A Huge Set of Built-In Functions

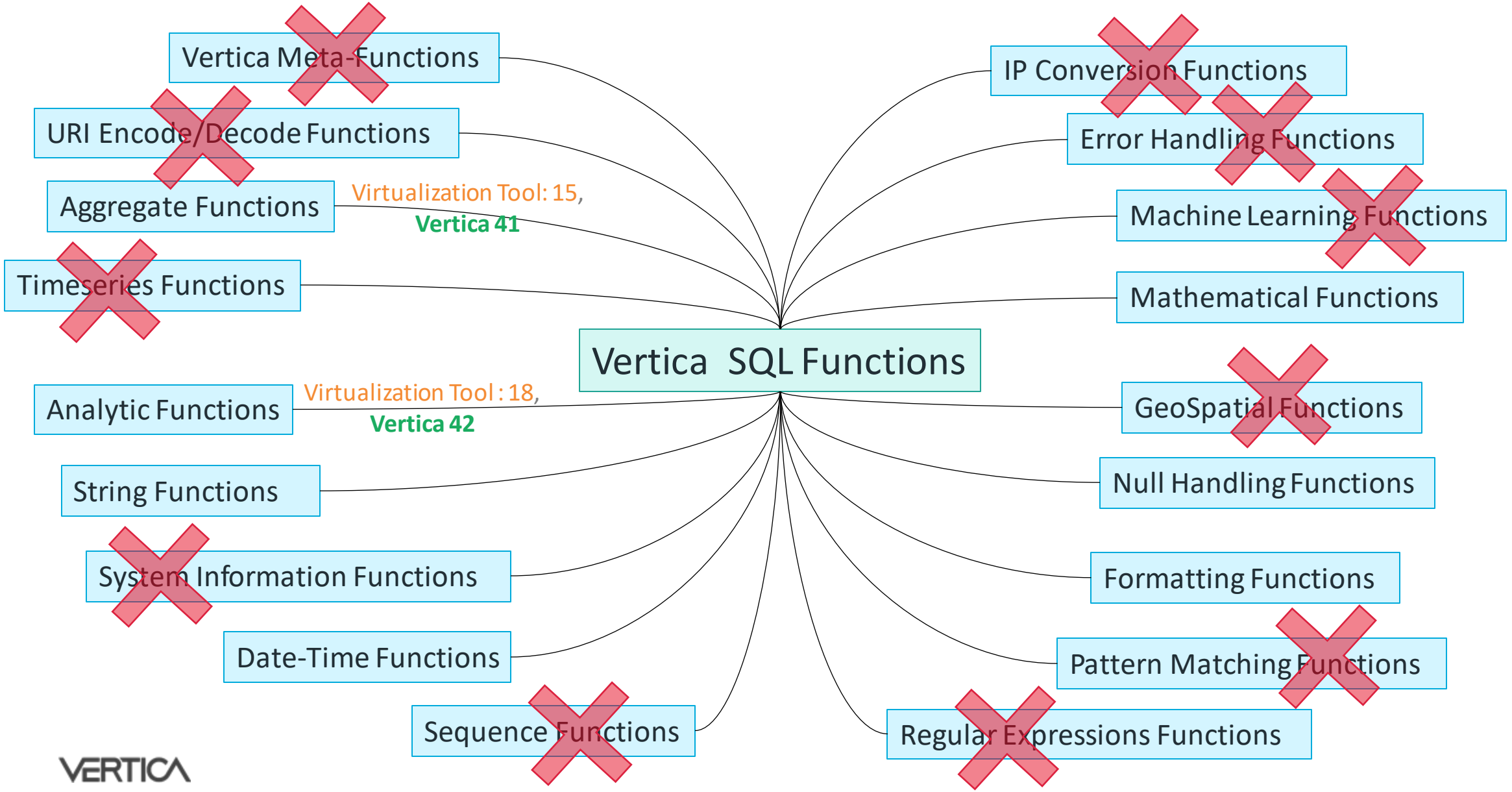


VERTICA

Vertica Functions Overview



Other Data Virtualization tools



VERTICA

Vertica Virtualization Layer – External Table

Vertica External Table

Map any data format

… everywhere

as if it were a normal Table !

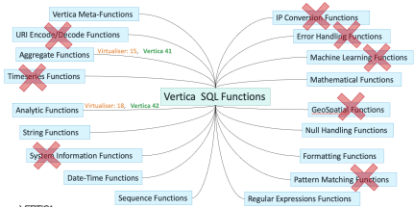
```
CREATE EXTERNAL TABLE public.epeople(
    id INTEGER,
    name VARCHAR
    ) AS COPY
        FROM '/data/epeople/*/*.parquet'
        PARQUET
        ;
```

When to Use External Tables

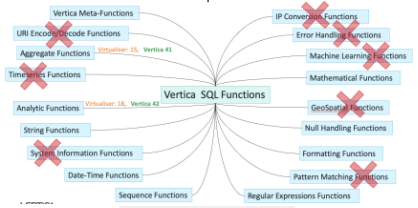
Input	Scan Processing / Reducing Unit	Output / Further Processing (Advanced Analytics / ML)	Method
Internal	Internal	Internal Vertica Engine	Vertica Table
Internal and External Source	Internal and External (Join filter / SIPS to Source) Table -> Other DBMS's Optimizer	Internal Vertica Engine	External Table: Federated Query
External Source	WHERE Filter in External Qry Table -> Other DBMS's Optimizer	Internal Vertica Engine	External Table: Federated Query
External Source	None or External (Part. Pruning, etc.) – *Except Flat Files*	Internal Vertica Engine	External Table: Other

Extrernal Table Types	Parser/Source	Efficient Method
Flat File based	Default CSV & Flex Parsers. No Filter	Make **internal**; Create Table and COPY
Parquet / Orc	Parquet/ORC source, via PARQUET/ORG keyword	Exploit filtering mechanisms & keep **external**
Federated Query Source	ODBC Source & ODBC Parser	Exploit the filter and column pruning pushdown mechanisms & keep **external**

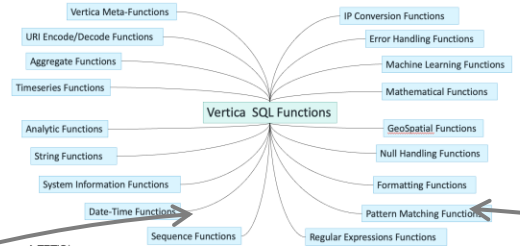
Why Pull Data Into Vertica



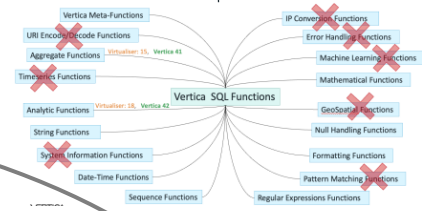
Data Virtualisation tool



HIVE



VERTICA



EXTERNAL DATABASES

Parquet

Apache orc

CSV

{JSON}

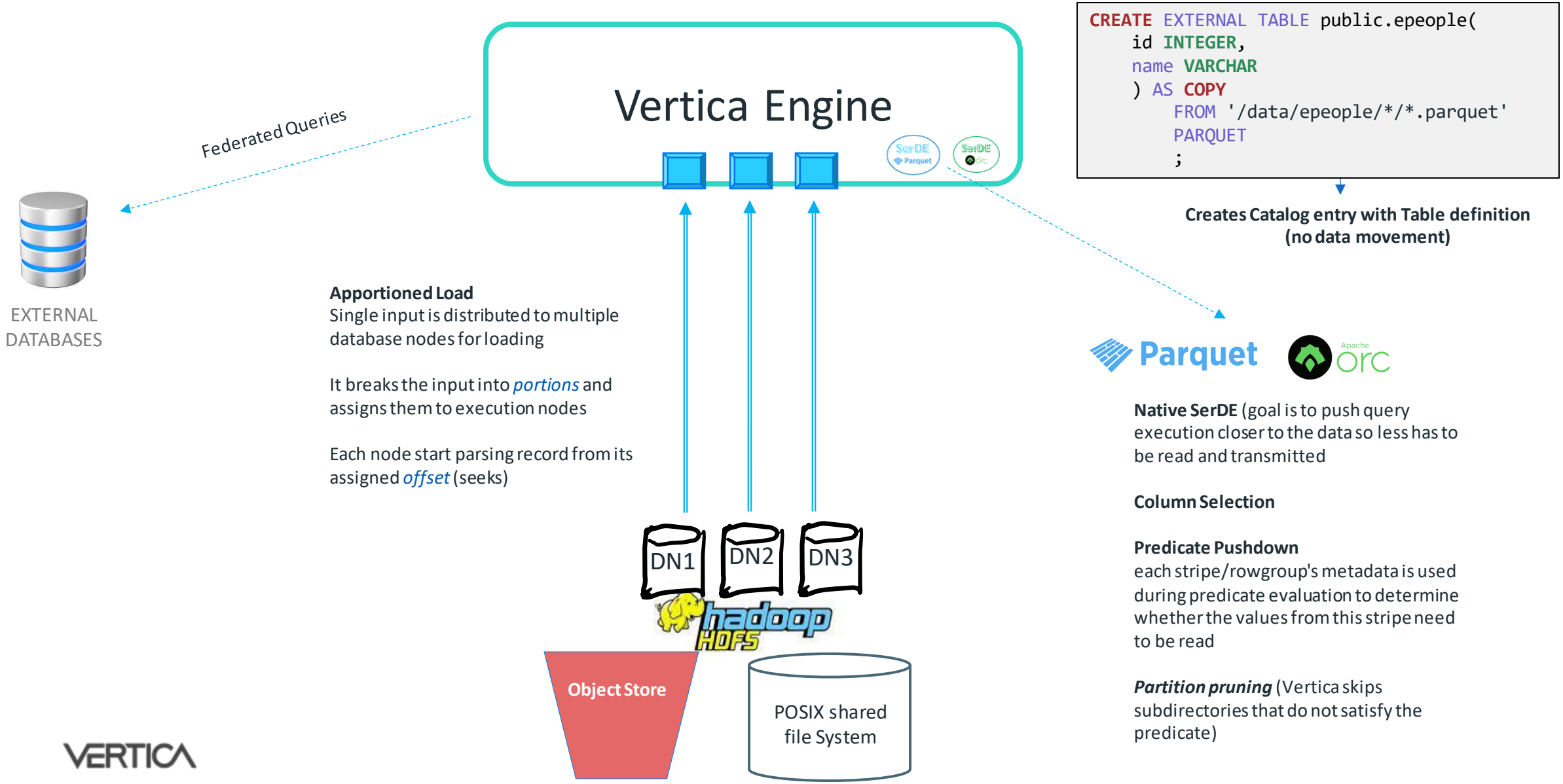
{ } API

OTHER FORMATS or INTERFACES

VERTICA

External Table
&
Complex Data Types

What happens when we query External Table




```
CREATE EXTERNAL TABLE public.epeople(
    id INTEGER,
    name VARCHAR
    ) AS COPY
        FROM '/data/epeople/*/*.parquet'
        PARQUET
        ;
```

Creates Catalog entry with Table definition (no data movement)

Parquet Apache orc

Native SerDE (goal is to push query execution closer to the data so less has to be read and transmitted

Column Selection

Predicate Pushdown
each stripe/rowgroup's metadata is used during predicate evaluation to determine whether the values from this stripe need to be read

Partition pruning (Vertica skips subdirectories that do not satisfy the predicate)

VERTICA

How to create external table

```
CREATE EXTERNAL TABLE
```

Define table COLUMNS as you would for a native table

```
public.epeople(
    id INTEGER,
    name VARCHAR ..
```

add FROM with external data location

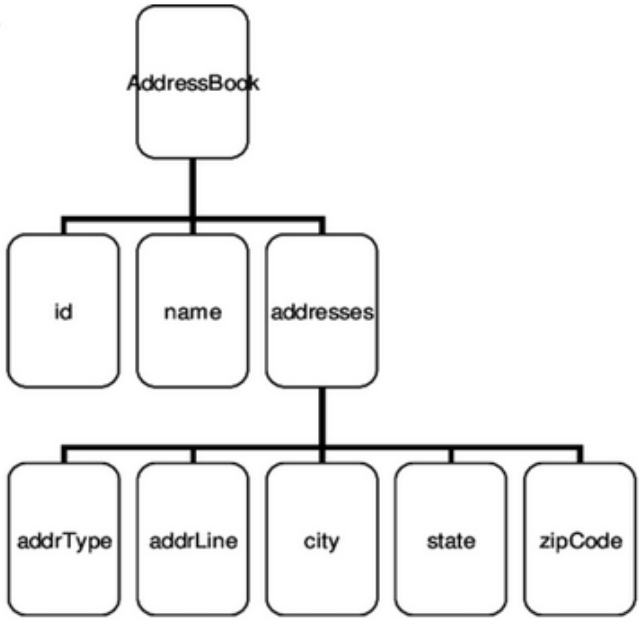
```
        FROM '/data/epeople/*/*.parquet'

        FROM 'S3://bucket/epeople..

        FROM 'gs .. .
```

And type (ORC, Parquet.. JSON…)

```
        PARQUET
        ;
```

→ ? How to describe columns for table with long structure and may be complex data type

```
<element name="AddressBook" type="tns:AddressBookType"/>
<complexType name="AddressBookType">
  <all>
    <element name="id" type="double/>
    <element name="name" type="string"/>
    <element name="addresses">
      <complexType>
        <all>
          <element name="address" type="tns:Address"
   minOccurs="0" maxOccurs="unbounded"/>
        </all>
      </complexType>
    </element>
  </all>
</complexType>
<complexType name="Address">
  <all>
    <element name="addrType" type="double"/>
    <element name="addressLine" type="string"/>
    <element name="city" type="string"/>
    <element name="state" type="string"/>
    <element name="zipCode" type="string"/>
  </all>
</complexType>
```



Automatically read external structure

Parquet Apache orc

INFER_EXTERNAL_TABLE_DDL

- Inspect a file in Parquet or ORC format and return the “CREATE” statement
- Fully define complex types, included nested types
- Eventually treat the column as LONG VARBINARY value
- Partition column = UNKNOWN data type
  - (Parquet/ORC contain insufficient info to define it)

Others:

- csv, delimited … `d2le` (https://github.com/marco-the-sane/d2l)
- JSON, CEF, AVRO … use built in parser

```
=> SELECT INFER_EXTERNAL_TABLE_DDL('/data/people/*.parquet'
        USING PARAMETERS format = 'parquet', table_name = 'employees');

                    INFER_EXTERNAL_TABLE_DDL
-----------------------------------------------------------------------
 create external table "employees"(
  "employeeID" int,
  "personal" Row(
    "name" varchar,
    "address" Row(
      "street" varchar,
      "city" UNKNOWN,
      "zipcode" int
    ),
    "taxID" int
  ),
  "department" varchar
 ) as copy from '/data/people/*/*.parquet' parquet;
(1 row)
```

same as struct

partitioning

```
WITH parser fjsonparser
```

VERTICA

Complex Data Type in Vertica native table

- New Complex Data Types introduced
  - **(ROW), ARRAY, SET**
  - **ARRAY** is a 1D unordered collection of elements, including duplicates (as-is as from the input)
  - **SET** is a 1D ordered and de-duplicated collection of elements. Implemented to optimize element look-up
  - when **ARRAY::SET**, Vertica sorts the values and removes duplicate elements, it can improve query checking for element presence

bounded

```
create table complex (id int,
                type_arrayc array[varchar, 10], type_setc set[varchar, 10] default type_arrayc::SET[varchar],
                type_arrayi array[int, 10],     type_seti set[int, 10] default type_arrayi::SET[int]
                );

insert into complex (type_arrayc, type_arrayi) values (ARRAY['a','b','b','d','c'], ARRAY[1,2,2,-1,5,5,3,0]);

select * from complex;
 id |      type_arrayc      |     type_setc     |    type_arrayi     |   type_seti
----+-----------------------+-------------------+--------------------+---------------
    | ["a","b","b","d","c"] | ["a","b","c","d"] | [1,2,2,-1,5,5,3,0] | [-1,0,1,2,3,5]
```


VERTICA

Why bounded Array ?

- Arrays and SET can be:
  - bounded, meaning they specify a maximum element count
  - or unbounded (have a maximum binary size)
    - Set explicitly
    - or defaulted
- BOUNDED Array/SET helps resource manager to execute efficiently because the bounds are known

```
dbadmin=> create table public.test1 ( a1 ARRAY[INT], a2 ARRAY[INT,200], a3 ARRAY[INT] (20000) );
CREATE TABLE

dbadmin=> \d public.test1;
                                    List of Fields by Tables
 Schema | Table | Column |        Type         | Size  | Default | Not Null | Primary Key | Foreign Key
--------+-------+--------+---------------------+-------+---------+----------+-------------+-------------
 public | test1 | a1     | array[int8](65000)  | 65000 |         | f        | f           |
 public | test1 | a2     | array[int8, 200]    |  1600 |         | f        | f           |
 public | test1 | a3     | array[int8](20000)  | 20000 |         | f        | f           |
(3 rows)
```

VERTICA

Vertica and … the Others

Feature	Vertica	Popular Cloud Solution	Popular SaaS Offering	Spark
SET Type	Yes	Yes	No	No
Full Support for Null/Empty Arrays	Yes	No	Yes	No
Bounded Arrays	Yes	No	No	No

VARCHAR vs Complex Data Types

- Customer has Wide (>1000 columns), Big (>160 Billion rows), Flat table containing lists of integer like in a text field: “13426,6732664,9892836,…” (hundreds of integers)
- Queries on this WBF table…
  - Always have a filter on date (`WHERE dt BETWEEN '20201101' AND '20210331'`)
  - Always have filters on list of integers text (`AND INSTR(vc,'994992')>0`)
- Scope of this test is to check how to optimize the table structure in order to optimize query elapsed

Test Environment

- Initial Table structure (public.at1):

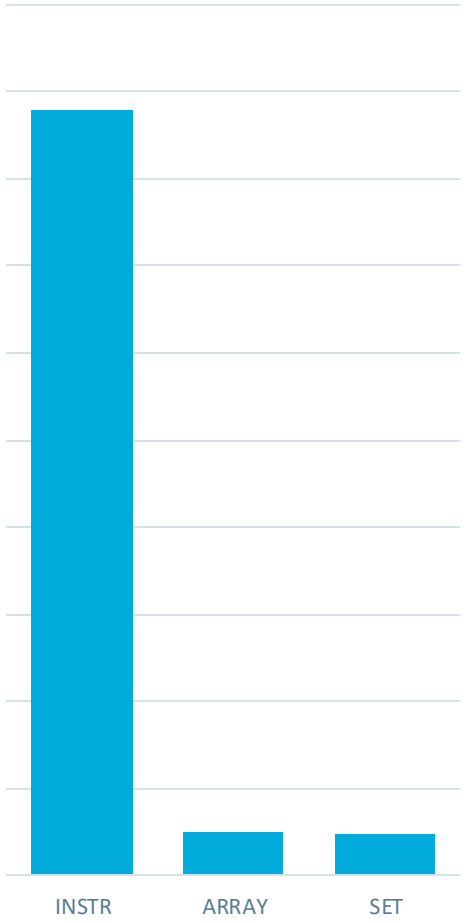
```
CREATE TABLE public.at1 (
    id INTEGER          NOT NULL ,
    vc VARCHAR(4096)    NOT NULL ,
    ar ARRAY[INT, 300] DEFAULT string_to_array('['||vc||']',  ',')::ARRAY[INT] ,
    se SET[INT, 300]    DEFAULT string_to_array('['||vc||']',  ',')::ARRAY[INT]::SET[INT]
   )
ORDER BY id
SEGMENTED BY HASH(id)
ALL NODES KSAFE;
```

- Has been loaded with 100ML rows generated this way:

```
$ nohup perl -e 'do{print $n++, "|", map{int(rand(2000000))-1000000,","}1..int(rand(300));
print int(rand(2000000))-1000000, "\n"} foreach 1..100000000' | vsql -AXntq -c "COPY
public.at1(id, vc) FROM STDIN DELIMITER '|' ABORT ON ERROR STREAM NAME 'INSTR_TEST'"
```

INSTR vs ARRAY vs SET – Query Performance

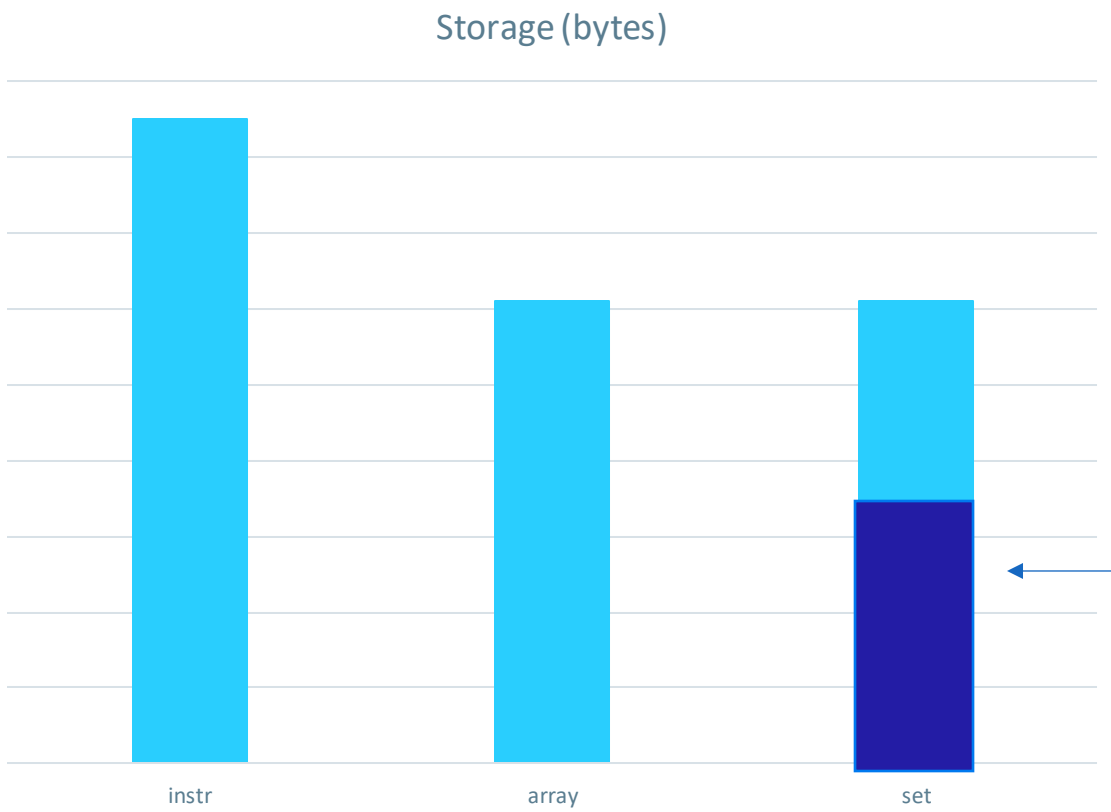
```
-- PERFORMANCE TEST (STRING SEARCH):
SELECT COUNT(*) FROM ( SELECT id FROM public.at1 WHERE INSTR(vc, '994992') > 0 ) x;
-- out  COUNT
-- out -------
-- out  15177
-- out (1 row)


-- PERFORMANCE TEST (ARRAY SEARCH):
SELECT COUNT(*) FROM ( SELECT id FROM public.at1 WHERE CONTAINS(ar, 994992)
                                                    OR CONTAINS(ar, -994992) ) x;
-- out  COUNT
-- out -------
-- out  15177
-- out (1 row)


-- PERFORMANCE TEST (SET SEARCH - MULTIPLE VALUES):
SELECT COUNT(*) FROM ( SELECT id FROM public.at1 WHERE CONTAINS(se, 994992)
                                                    OR CONTAINS(se, -994992) ) x;
-- out  COUNT
-- out -------
-- out  15177
-- out (1 row)
```



ARRAY/SET ≈ 18 times faster than INSTR

VERTICA

Storage Requirements



- ARRAY/SET need 29% less storage resources than VARCHAR
- SET needs 1% less storage than ARRAY
  - (Influenced by redundancies)
  - Random generated values
  - If the collection has 300 random elements

Flattening vs. Complex Data Types

- We construct a micro-benchmark to evaluate flattening the data vs. using a complex data type
- The micro-benchmark consists of user ids and corresponding search words searched by the user
  - The data set consists of 1,000,000 users each with up to 1000 distinct search words
- The table schemas used are below

```
CREATE TABLE words_flat   (userName VARCHAR, words VARCHAR);

CREATE TABLE words_array  (userName VARCHAR, words ARRAY[VARCHAR, 1000]);

CREATE TABLE words_set    (userName VARCHAR, words SET[VARCHAR, 1000]);
```

Flattening vs. Complex Data Types

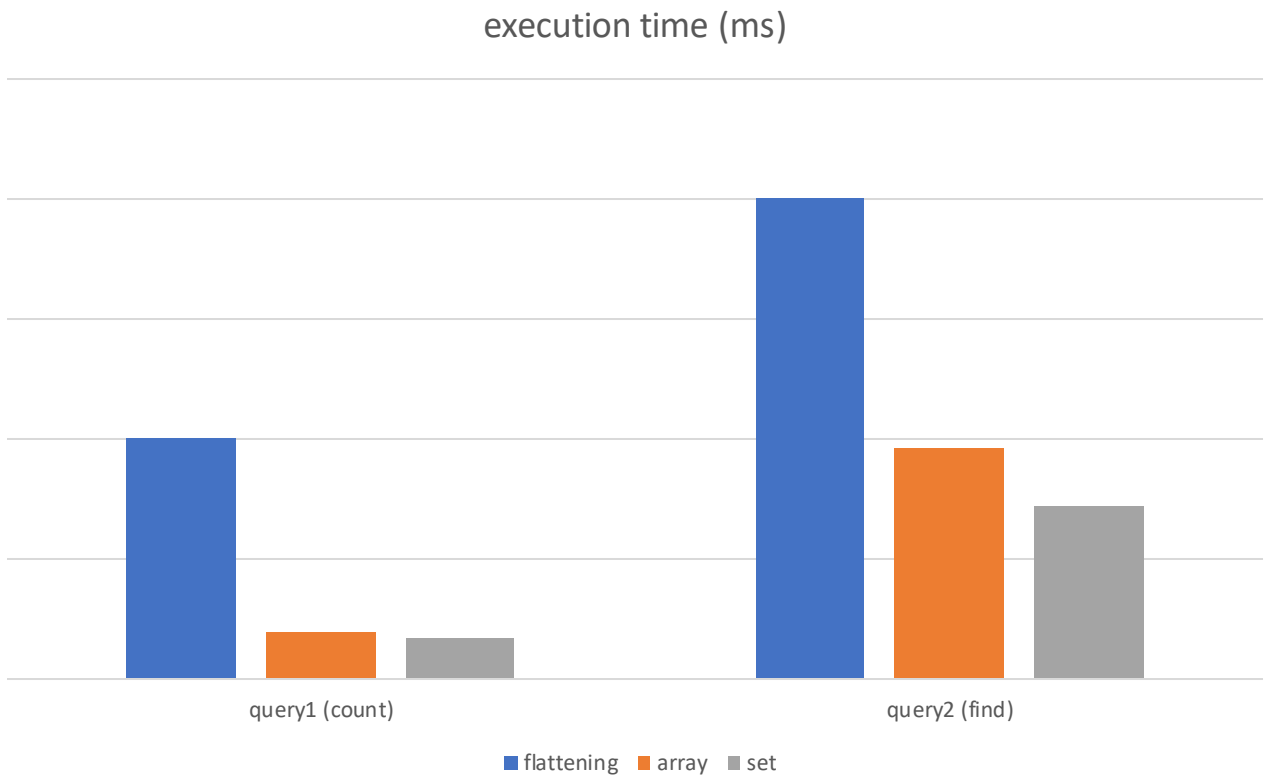
- The queries used:
  - Query 1: Count the number of search words used by each user:

```
SELECT userName, count(words) FROM words_flat GROUP BY userName;
SELECT userName, apply_count(words) FROM words_array;
SELECT userName, apply_count(words) FROM words_set;
```

  - Query 2: Find users that used a specific search word

```
SELECT userName FROM words_flat  WHERE words = 'XBox';
SELECT userName FROM words_array WHERE contains(words,'XBox');
SELECT userName FROM words_set   WHERE contains(words,'XBox');
```

Execution time

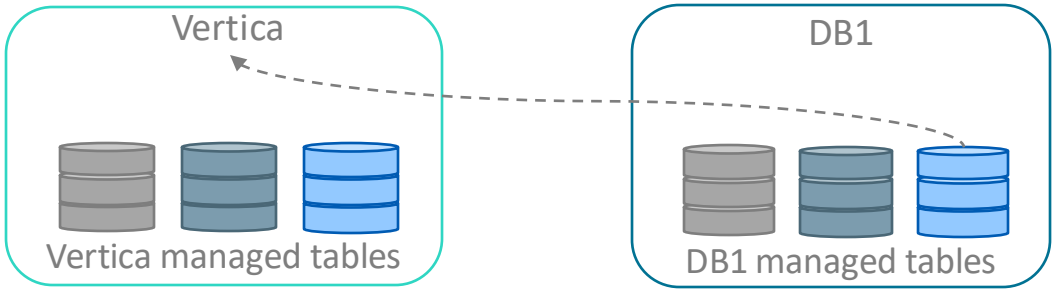


- In COUNT(*):
  - ARRAY approach is 5+ times faster
  - SET is 6+ times faster
- In find:
  - ARRAY is 2+ times faster than flattening approach
  - SET is ≈ 3 times faster

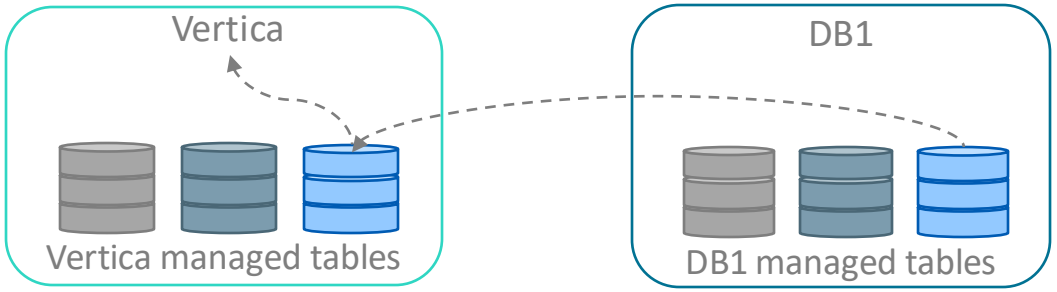
Vertica Federated Queries

Vertica Federated Queries - Scope

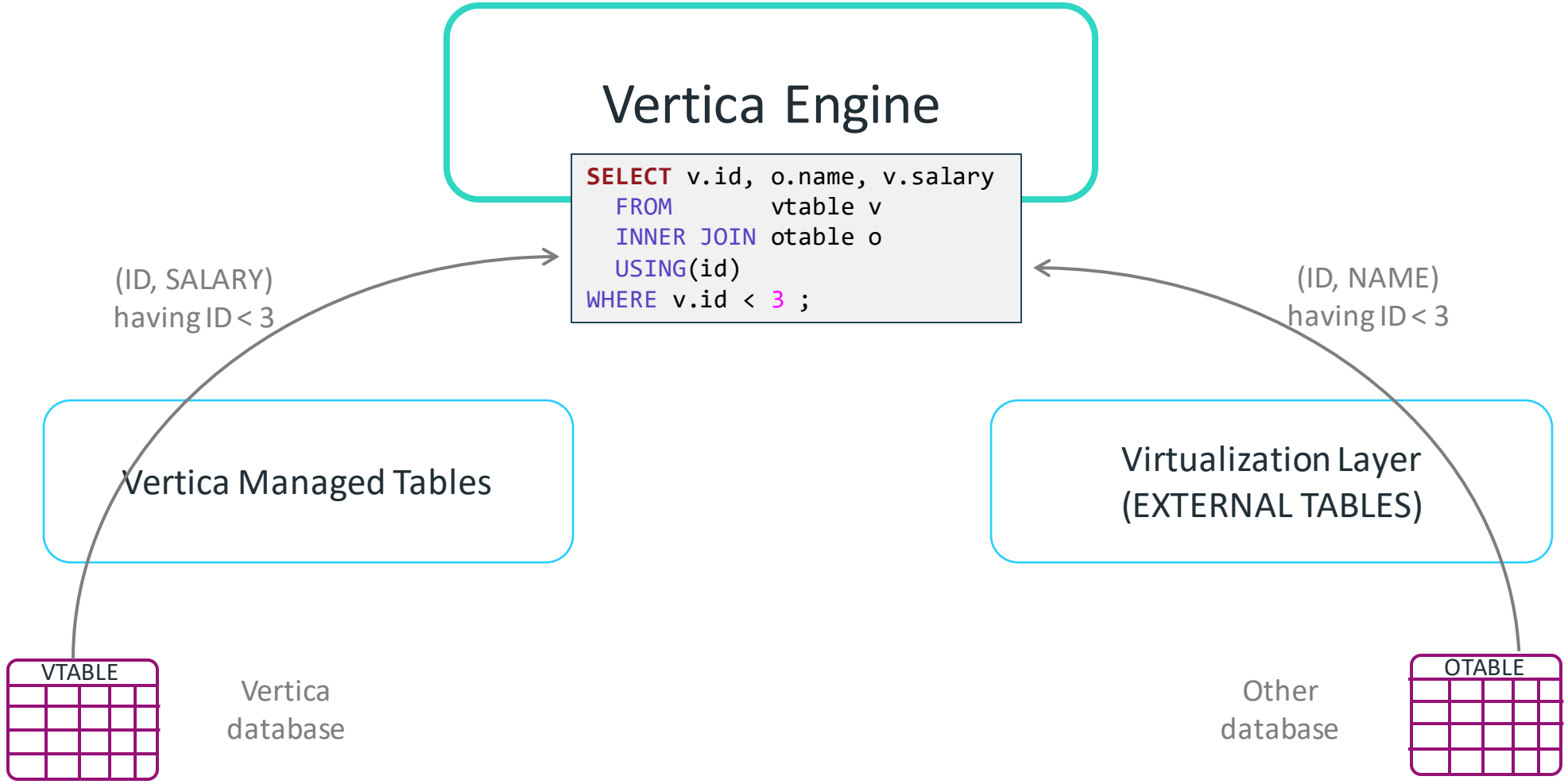
- Run queries against other databases



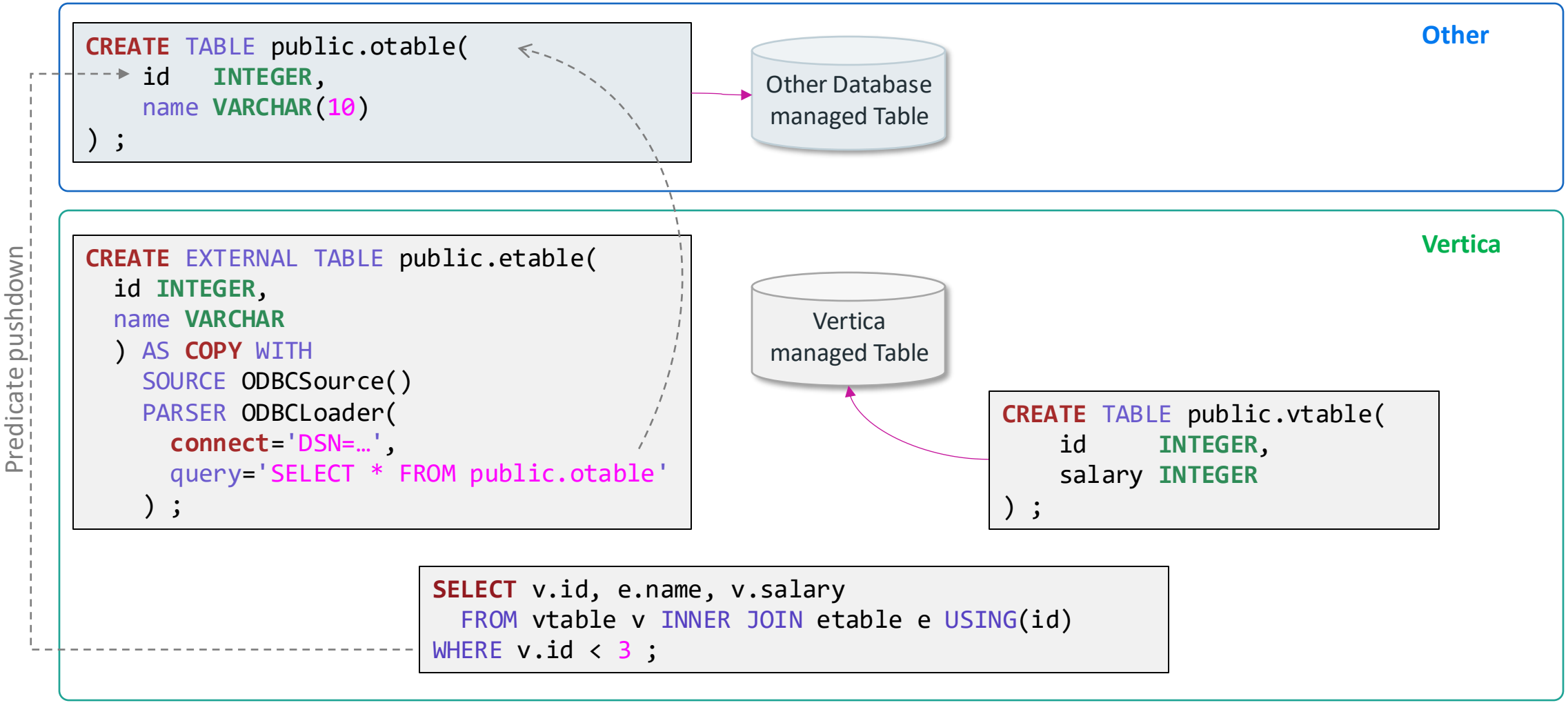
- Join Vertica and non-Vertica managed tables



Example


Vertica Engine
SELECT v.id, o.name, v.salary
FROM vtable v
INNER JOIN otable o
USING(id)
WHERE v.id < 3 ;
(ID, SALARY)
having ID < 3
(ID, NAME)
having ID < 3
Vertica Managed Tables
Virtualization Layer
(EXTERNAL TABLES)
VTABLE
Vertica
database
Other
database
OTABLE

How it works…



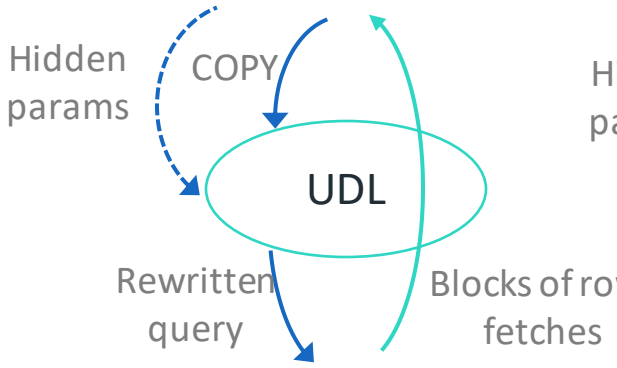
https://github.com/vertica/Vertica-Extension-Packages/tree/master/odbc_loader_package

Vertica Federated Queries and the time axis

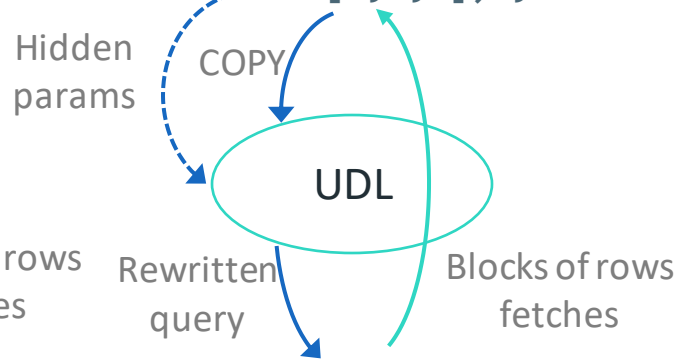
VERTICA DB

```
CREATE EXTERNAL TABLE etable
   AS COPY (…
      query = "SELECT * FROM otable"
   ) ;
```

```
SELECT C4, C5, C6
   FROM etable
   WHERE C1 < 3 ;
```




```
SELECT C1::INTEGER
   FROM etable
   WHERE C4 = ANY(
    ARRAY[1,2,3]) ;
```



OTHER DB

Nothing

```
SELECT C4, C5, C6 FROM
 (SELECT * FROM otable) s
WHERE C1 < 3
```

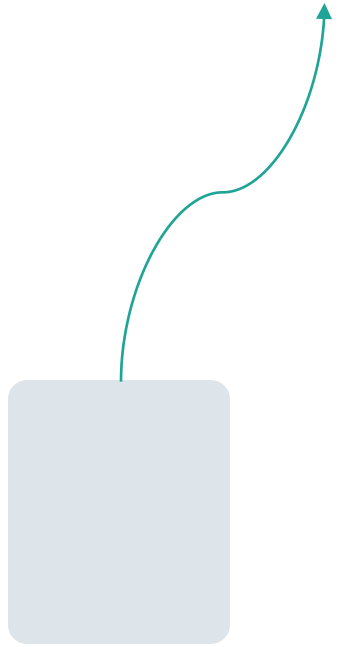
```
SELECT C1 FROM
(SELECT * FROM otable) s
WHERE C4 IN (1,2,3)
```

What happens when we Query External Tables

Other DB: otable

C0	C1	C2	C3	C4	C5	C6	C7	C8	C9

VERTICA: SELECT … FROM etable



VERTICA

Query conversion examples

```
CREATE TABLE public.mypeople(
    id              INTEGER  ,
    salary          NUMERIC(8,2)

;
```

```
CREATE EXTERNAL TABLE public.ext_mypeople(
    id              INTEGER  ,
    name            VARCHAR(10),
    gender          CHAR(1),
    bdate           DATE
) AS COPY WITH SOURCE ODBCSource() PARSER ODBCLoader(
        connect='DSN=mmf', query='SELECT * FROM mmf.mypeople' ) ;
```

Query in Vertica	Query against the other database
`SELECT COUNT(*) FROM ext_mypeople`	`SELECT id, NULL, NULL, NULL FROM (SELECT * FROM mmf.mypeople) sq`
`SELECT COUNT(*) FROM ext_mypeople WHERE gender = 'F'`	`SELECT NULL, NULL, gender, NULL FROM (SELECT * FROM mmf.mypeople) sq WHERE (gender = 'F')`
`SELECT v.id, m.name, v.salary FROM mypeople v INNER JOIN ext_mypeople m USING(id) WHERE v.id > 3`	`SELECT id, name, NULL, NULL FROM (SELECT * FROM mmf.mypeople) sq WHERE (id > 3)`

Other implemented features…

- The loader UDL converts some Vertica-specific syntax:
  - "~~" will be translated to `LIKE`,
  - "`ANY(ARRAY[1,2,3])`" will became `IN(1,2,3))`
  - `::<data type>` CAST removed from select list
- Configure the "rowset" (default 100) to define number of rows fetched for each SQLFetch() iteration
- Possibility to *override* original query at run-time using SESSION PARAMETERS

```
ALTER SESSION SET UDPARAMETER FOR ODBCLoaderLib override_query ='<new query here>';
```

- Possibility to switch on/off rows and/or columns filtering via SESSION PARAMETERS

```
ALTER SESSION SET UDPARAMETER FOR ODBCLoaderLib src_cfilter = false ;
```

```
ALTER SESSION SET UDPARAMETER FOR ODBCLoaderLib src_rfilter = false ;
```

The others… (Popular Cloud-only Database #1)

- Step 1: Run the query in the external database
- Step 2: Export the result set to S3
- Step 3: run the query in “Popular Cloud-only Database 1) reading from S3
- In Vertica we never land a single byte to disk…

The others… (Popular Cloud-only Database #2)

- "Federated Queries" against MySQL & PostgreSQL instances in Cloud SQL
- Very involuted approach
- Data Type mapping at query run time

```
SELECT c.customer_id, c.name, SUM(t.amount) AS total_revenue,
rq.first_order_date
FROM customers AS c
INNER JOIN transaction_fact AS t ON c.customer_id = t.customer_id
LEFT OUTER JOIN EXTERNAL_QUERY(
  'connection_id',
  '''SELECT customer_id, MIN(order_date) AS first_order_date
  FROM orders
  GROUP BY customer_id''') AS rq ON rq.customer_id = c.customer_id
GROUP BY c.customer_id, c.name, rq.first_order_date;
```

The others… (more primitive solutions)

- Some others pretend you to create “EXTERNAL” tables with matching column names
- No predicate push-down
- Data Type mapping at query run time

Conclusions

- Vertica was already able to access data in different storage locations and formats
- We are now able access data managed by other databases in a much more efficient way
- Remember… query are executed IN Vertica
- Vertica has complex native data type
- Numbers are very good and customers are very happy:

… works like a charm, WOW…

…This is an incredible feature that doesn't exist for any competing DB platform that once again shows Vertica's advantage…

VERTICA

Thank you

Vertica Community Edition + Docker

www.vertica.com/trial
https://hub.docker.com/r/verticadocker/vertica-ce

- Deploy Vertica on up to 3 nodes
- Store up to 1TB of structured and semi-structured data
- Use Vertica with no time limits or license requirements