Cloudera Data Warehouse Private Cloud 1.5.0

Unified Analytics overview

Date published: 2023-01-24
Date modified: 2023-01-25

https://docs.cloudera.com/

Legal Notice

Contents

Unified Analytics overview

You can take advantage of SQL engine enhancements in the Cloudera by using Unified Analytics. Unified Analytics includes semantics commonality, backward compatibility, and optimizations.

Important: The Unified Analytics feature in the Cloudera Data Warehouse on premise is now deprecated. It is no longer supported in the user interface, but you can still manage the existing instances using the Cloudera CLI.

This release of Unified Analytics is the first product of a long-term vision that brings SQL equivalency without syntax changes to Cloudera Data Warehouse SQL engines. Unified Analytics is available in the following Cloudera environments:

- Red Hat OpenShift
- Embedded Container Service (ECS)

Unified Analytics SQL engines support SQL:2016 syntax and semantics, including the following key operations:

- Automatic query rewrites to use materialized views
- Command-line materialized view recommender
- DataSketches functions and rewrites
- Ranger column masking and row filtering
- Query results cache
- SQL set operations and grouping sets
- Atlas integration
- Extensive subquery support
- Advanced join reordering with bushy plans generation
- Integrity constraints-based rewritings
- User defined functions (UDFs) in Hive
- Other extensions to query optimization, such as column pruning, sort/limit merge and pushdown

Unified Analytics also brings significant optimization equivalency to the SQL engines, unifying common techniques such as subquery processing, join ordering and materialized views.

Lexical conventions

The default behavior for SQL queries in Unified Analytics is to use single quotation marks for the literals. The ANSI SQL standard is to use single quotation marks for string literals. If a single quotation mark or special character appears within that string literal, it needs to be escaped. For example:

```
INSERT INTO MOVIES_INFO VALUES
(1,cast('Toy Story (1995)' as varchar(50)), 'Animation|Children\'s|Comedy'),
...
```

You can change the SQL behavior. In your Virtual Warehouse, click Edit Configurations HiveServer2 , and add the hive.support.quoted.identifiers property. Set this property to one of the following values:

- none

  Quotation of identifiers and special characters in identifiers are not allowed, but regular expressions in backticks are supported for column names.
- column

  Use the backtick character to enclose identifiers having special characters. `col1`. Use single quotation marks to enclose string literals, for example: 'value'. Double quotation marks are also accepted, but not recommended.

- standard (default)

  SQL standard way to enclose identifiers. Use double quotation marks to enclose identifiers having special characters "col1" and single quotation marks for string literals, for example 'value'.

Limitations

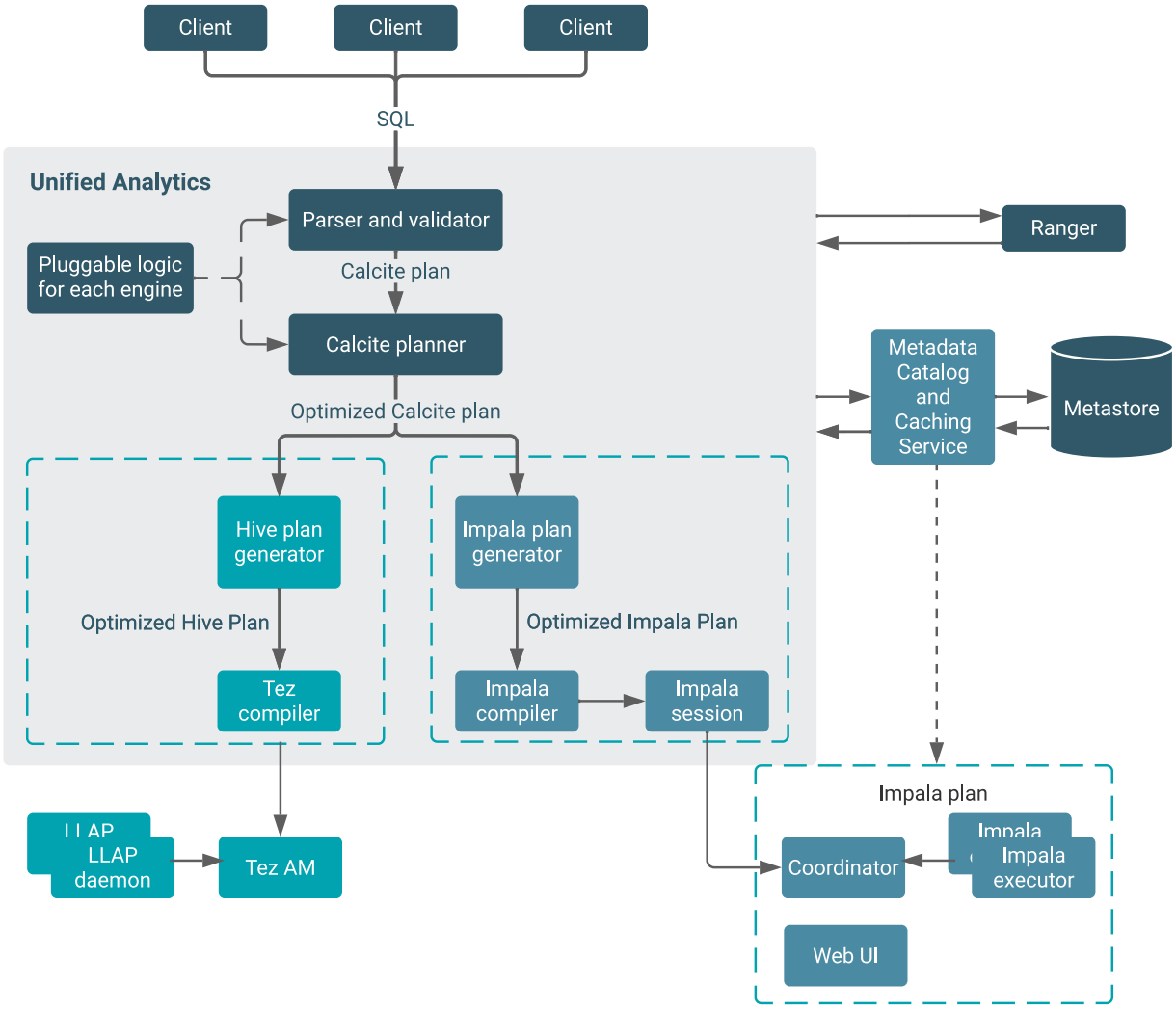
- Unified Analytics does not support left/right ANTI JOIN syntax.
- Unified Analytics does not support complex types - ARRAY, STRUCT, MAP.

Unified Analytics architecture

The architecture of Unified Analytics supports Cloudera's long-term vision to make Impala and Hive engines equivalent in SQL functionality without syntax changes. You learn the part that each major Unified Analytics component in Cloudera Data Warehouse plays in improving your analytics experience.

The architecture brings significant optimization equivalency to the both SQL engines. Each engine has its own style of doing optimizations within the query processor, but there are common techniques such as subquery processing, join ordering and materialized views, which the architecture unifies. The architecture is compatible with the HiveServer (HS2) protocol.

The following diagram shows the Unified Analytics architecture.



The diagram includes the following components:

Clients

Connect to the Unified Analytics endpoint parser and validator.

Ranger

Secures data and supports column masking and row filtering in both engines; whereas, only column masking is available to the un-unified Impala.

Metastore, metadata catalog and caching service

Shared by both engines

Calcite planner

Receives SQL queries from the parser/validator, and generates a plan for Hive or Impala.

Optimized Calcite plan

Generated for Hive or Impala.

Tez compiler

Hands off Tez tasks that run on LLAP daemons to the Tez Application Manager.

Impala session

Connects to the coordinator, runs on executors, and the Web UI shows runtime query profiles for diagnosing problems.

Tez AM and LLAP daemons

The Tez Application Manager (AM) runs the Hive job in LLAP mode, obtaining free threads from LLAP daemons to quickly execute fragments of the plan.

Impala plan

Capabilities and strengths of Impala, such as different types of join algorithms, different types of distribution, broadcast and partitioning for example, generate an Impala specific plan if your client connected to the Impala Virtual Warehouse.

Calcite provides SQL query planning capabilities to both engines. This represents no change to Hive, but adds many new enhancements to Impala. Because the Calcite logic is pluggable for each engine, Unified Analytics can support engine differences, such as type checking, and data and timestamp functions. The pluggable logic based on engine semantics generates appropriate expressions and types. You realize the following benefits of this logic:

- Function validation
- Type system implementation
- Expressions simplification
- Static partition pruning

For example, multiplying decimals returns results in Hive output different from the Impala output. This architecture makes backward compatibility and forward commonality possible. The pluggable framework expedites adding new Calcite transformation rules for delivering new cost-based optimizations.

Partition pruning and predicate push-down are supported by Hive and Impala. Unified Analytics seamlessly handles partition pruning semantics, which can differ because you can have expressions in the pruning predicates. Support for join reordering, materialized view rewriting, and other capabilities, formerly only in Hive, are available in Impala. The query results cache supported in Hive 3 is also now available to Impala.

ORC vs Parquet formats

The differences between Optimized Row Columnar (ORC) file format for storing data in SQL engines are important to understand. Query performance improves when you use the appropriate format for your application.

ORC and Parquet capabilities comparison

The following table compares SQL engine support for ORC and Parquet.

Table 1:

Capability	Cloudera Data Warehouse	ORC	Parquet	SQL Engine
Read non-transactional data	Apache Hive	#	#	Hive
Read non-transactional data	Apache Impala	#	#	Impala
Read/Write Full ACID tables	Apache Hive	#		Hive
Read Full ACID tables	Apache Impala	#		Impala
Read Insert-only managed tables	Apache Impala	#	#	Impala
Column index	Apache Hive	#	#	Hive
Column index	Apache Impala		#	Impala
CBO uses column metadata	Apache Hive	#		Hive

Capability	Cloudera Data Warehouse	ORC	Parquet	SQL Engine
Recommended format	Apache Hive	#		Hive
Recommended format	Apache Impala		#	Impala
Vectorized reader	Apache Hive	#	#	Hive
Read complex types	Apache Impala	#	#	Impala
Read/write complex types	Apache Hive	#	#	Hive

Supported user defined functions

Unified Analytics supports user defined functions in both SQL engines. You learn which types of UDFs are supported and see usage examples.

You can use the following types of UDFs:

- Hive Java UDF

```
create function hive_f(int) returns int LOCATION
S3a://my-bucket/path.jar SYMBOL='my_java_func'
```

- Native C++ UDFs created in Impala

```
create function native_fn(int) returns
int LOCATION S3a://my-bucket/path.so SYMBOL='my_cpp_func'
```

- Native C++ UDAs created in Impala

```
create aggregate function agg_fn(int)
returns int LOCATION S3a://my-bucket/path.so SYMBOL='agg_func'
```

After creating the C++ functions, you must run RELOAD FUNCTION or RELOAD FUNCTIONS to load UDFs you created from Impala into HiveServer (HS2). For example:

```
RELOAD FUNCTIONS
```

For example:

```
select native_fn(i1) from tbl1;
| _c0  |
3
3

select agg_fn(i1) from tbl1;
| _c0  |
2

select hive_f(s1) from tbl1;
| _c0  |
119
104
```