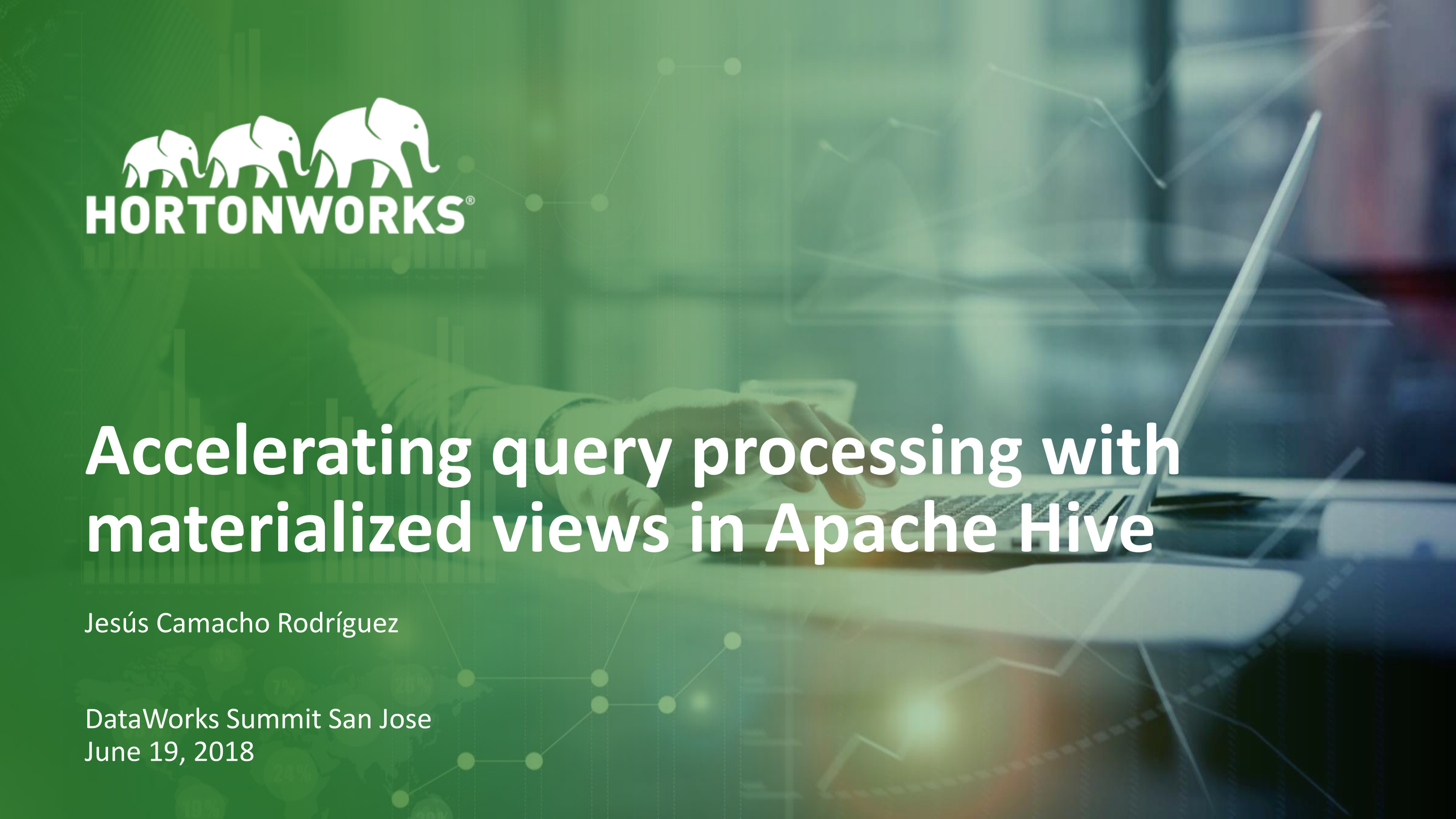


Accelerating query processing with materialized views in Apache Hive

Jesús Camacho Rodríguez

DataWorks Summit San Jose
June 19, 2018

Apache Hive

- Initial use case: batch processing
  - Read-only data
  - HiveQL (SQL-like query language)
  - MapReduce
- Effort to take Hive beyond its batch processing roots
  - Started in Apache Hive 0.10.0 (January 2013)
  - Latest release: Apache Hive 3.0 (May 2018)
- Extensive renovation to improve **three different axes**
  - Latency: allow interactive and sub-second queries
  - Scalability: from TB to PB of data
  - SQL support: move from HiveQL to SQL standard

HORTONWORKS

Apache Hive

Important internals improvements

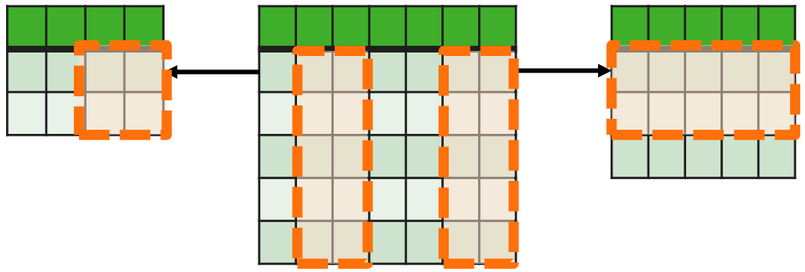
- Multiple execution engines: Apache Tez and Apache Spark
- More efficient join execution algorithms
- Vectorized query execution
  - Integration with columnar storage formats: Apache ORC, Apache Parquet
- LLAP (*Live Long and Process*)
  - Persistent deamons for low-latency queries
- Rule-based and cost-based optimizer
  - Better statistics
- Tighter integration with other data processing systems: Druid

HORTONWORKS

Accelerating query processing

Optimization based on access patterns

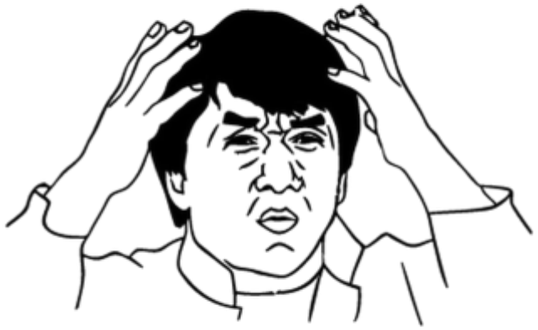
- Change data physical properties (distribute, sort)
- Filter rows
- Denormalize
- Preaggregate



HORTONWORKS

Accelerating query processing

Currently, Hive users have to do it manually



Optimization based on access patterns

- Establish relationship between original and new tables
  - Has a similar table already been created?
- Rewrite your queries to use new tables
  - What happens when access patterns change?
- Maintain your new tables when original tables change
  - Do I have to fully rebuild new tables?

HORTONWORKS

Materialized views

- A **materialized view** is an entity that contains the result of a evaluating a query
  - Important property → Awareness of the materialized view definition semantics
  - Optimizer can exploit them for automatic query rewriting
  - System can handle maintenance of the materialized views

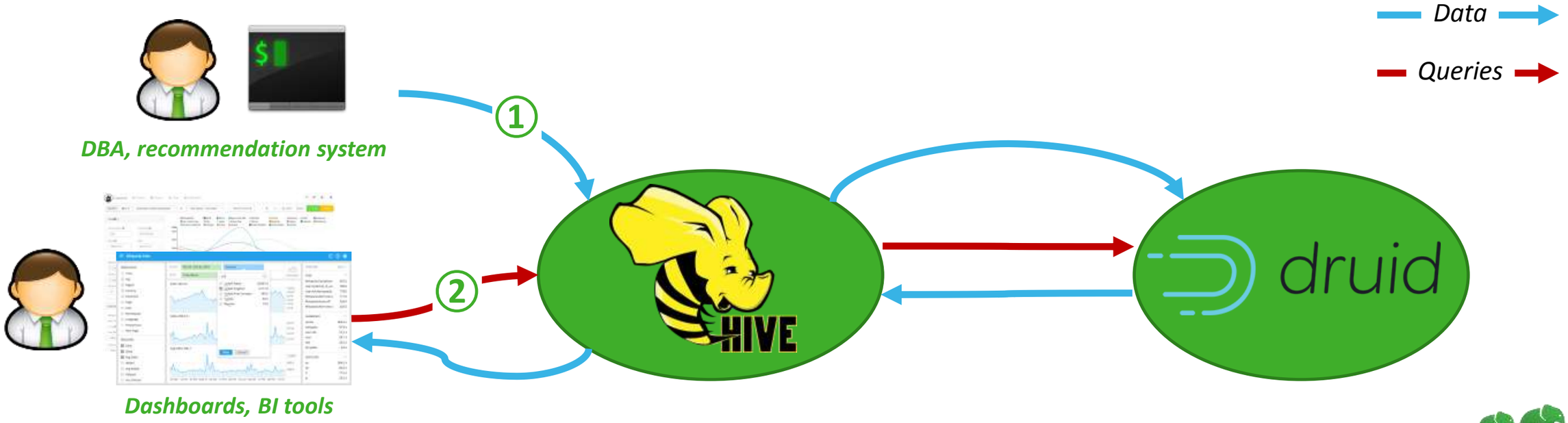
- Generally, materializations can be created in different forms depending on the scope
  - DBA writes “CREATE MATERIALIZED VIEW” statement
  - Daemon creates materialized view based on recent query activity
  - Cached result of previous similar query
  - Query factorization identifies common pieces within a single query

HORTONWORKS

Possible workflow

```
CREATE MATERIALIZED VIEW `ssb_mv`
STORED AS 'org.apache.hadoop.hive.druid.DruidStorageHandler'
ENABLE REWRITE
AS
<query>;
```

1. Create materialized view using Hive tables
   - Stored by Hive or Druid
2. User or dashboard sends queries to Hive
   - Hive rewrites queries using available materialized views
   - Execute rewitten query



Materialized views in Apache Hive

- First implementation is part of **Apache Hive 3.0**
- Multiple storage options: Hive, Druid
- Automatic rewriting of incoming queries to use materialized views
- Efficient view maintenance
  - Incremental refresh
- Multiple options to control materialized views lifecycle

HORTONWORKS

Management of materialized views in Hive

HORTONWORKS

Materialized view creation

- *CREATE MATERIALIZED VIEW* statement

```
CREATE MATERIALIZED VIEW [IF NOT EXISTS] [db_name.]materialized_view_name
  [DISABLE REWRITE]
  [COMMENT materialized_view_comment]
  [
   [ROW FORMAT row_format]
   [STORED AS file_format]
     | STORED BY 'storage.handler.class.name' [WITH SERDEPROPERTIES (...)]
  ]
  [LOCATION hdfs_path]
  [TBLPROPERTIES (property_name=property_value, ...)]
AS
<query>;
```

⇢ Supports custom table properties, storage format, etc.

HORTONWORKS

Materialized view creation (stored in Druid)

- *CREATE MATERIALIZED VIEW* statement

```
CREATE MATERIALIZED VIEW druid_wiki_mv
STORED AS 'org.apache.hadoop.hive.druid.DruidStorageHandler'
AS
SELECT __time, page, user, c_added, c_removed
FROM src;
```

Hive materialized view name
Hive storage handler classname

HORTONWORKS

Other operations for materialized view management

```
DROP MATERIALIZED VIEW [db_name.]materialized_view_name;
SHOW MATERIALIZED VIEWS [IN database_name] ['identifier_with_wildcards'];
DESCRIBE [EXTENDED | FORMATTED] [db_name.]materialized_view_name;
```

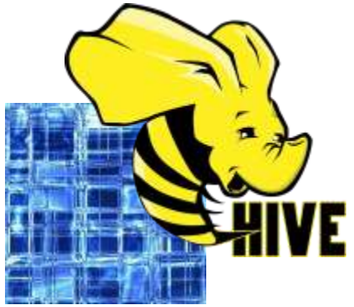
- More operations to be added and extended

HORTONWORKS

Materialized view-based query rewriting

HORTONWORKS

Materialized view-based rewriting algorithm



- Automatically rewrite incoming queries using materialized views
  - Optimizer exploits materialized view definition semantics
- Built on the ideas presented in [GL01] using **Apache Calcite**
  - Supports queries containing TableScan, Project, Filter, Join, Aggregate operators
- Includes some extensions
  - Generation of additional rewritings without needing to do join permutation
  - Partial rewritings using union operators
- More information about the rewriting coverage
  - http://calcite.apache.org/docs/materialized_views#rewriting-using-plan-structural-information

[GL01] Jonathan Goldstein and Per-åke Larson. Optimizing queries using materialized views: A practical, scalable solution. In Proc. ACM SIGMOD Conf., 2001.

HORTONWORKS

Enable materialized view-based rewriting

- Global property to enable materialized view rewriting for queries

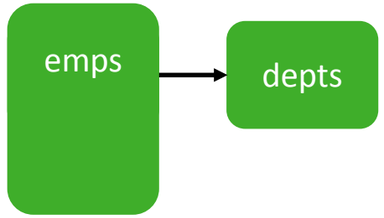
```
SET hive.materializedview.rewriting=true;
```

- User can selectively use enable/disable materialized views for rewriting
- Materialized views are enabled by default for rewriting
- Behavior can be altered after materialized view has been created

```
ALTER MATERIALIZED VIEW [db_name.]materialized_view_name ENABLE|DISABLE REWRITE;
```


HORTONWORKS

Materialized view-based rewriting (example)



- Materialized view definition

Employees that were hired after 2016

```
CREATE MATERIALIZED VIEW mv
AS
SELECT empid, deptname, hire_date
FROM emps JOIN depts
  ON (emps.deptno = depts.deptno)
WHERE hire_date >= '2016-01-01';
```

- Query

Employees that were hired last quarter

```
SELECT empid, deptname
FROM emps JOIN depts
  ON (emps.deptno = depts.deptno)
WHERE hire_date >= '2018-01-01'
    AND hire_date <= '2018-03-31';
```

- Materialized view-based rewriting

```
SELECT empid, deptname
FROM mv
WHERE hire_date >= '2018-01-01'
    AND hire_date <= '2018-03-31';
```

mv contents

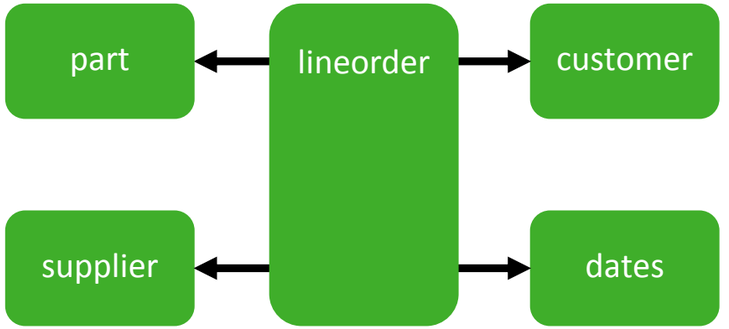
empid	depname	hire_date
10001	IT	2016-03-01
10002	IT	2017-01-02
10003	HR	2017-07-01
10004	**Finance**	**2018-01-15**
10005	**HR**	**2018-02-02**

Query results

empid	depname
10004	Finance
10005	HR

HORTONWORKS

Materialized view-based rewriting (example 2)



- Materialized view definition

```
CREATE MATERIALIZED VIEW mv AS
SELECT <dims>,
   lo_revenue,
   lo_extendedprice * lo_discount AS d_price,
   lo_revenue - lo_supplycost
FROM
   customer, dates, lineorder, part, supplier
WHERE
   lo_orderdate = d_datekey
   and lo_partkey = p_partkey
   and lo_suppkey = s_suppkey
   and lo_custkey = c_custkey;
```

- Query

```
SELECT sum(lo_extendedprice * lo_discount)
FROM
   lineorder, dates
WHERE
   lo_orderdate = d_datekey
   and d_year = 2013
   and lo_discount between 1 and 3;
```

- Materialized view-based rewriting

```
SELECT SUM(d_price)
FROM mv
WHERE
   d_year = 2013
   and lo_discount between 1 and 3;
```

Exploit SQL PK-FK and NOT NULL constraints

mv contents

d_year	lo_discount	<dims>	d_price
2013	**2**	**…**	**7.55**
2014	4	…	432.60
2013	**2**	**…**	**34.45**
2012	2	…	2.05
…	…	…	…

Query results

sum
42.0
…

HORTONWORKS

Materialized view-based rewriting (example 3)

wiki

- Materialized view definition

```
CREATE MATERIALIZED VIEW mv AS
SELECT floor(time to minute), page,
    SUM(added) AS c_added,
    SUM(removed) AS c_rmv
FROM wiki
GROUP BY floor(time to minute), page;
```

- Query

```
SELECT floor(time to month),
    SUM(added) AS c_added
FROM wiki
GROUP BY floor(time to month);
```

- Materialized view-based rewriting

```
SELECT floor(time to month),
    SUM(c_added) as c_added
FROM mv
GROUP BY floor(time to month);
```

mv contents

__time	page	c_added	c_rmv
2011-01-01 01:05:00	Justin	1800	25
2011-01-20 19:00:00	Justin	2912	42
2011-01-01 11:06:00	Ke$ha	1953	17
2011-02-02 13:15:00	Ke$ha	3194	170
2011-01-02 18:00:00	Miley	2232	34

Query results

__time	c_added
2011-01-01 00:00:00	8897
2011-02-01 00:00:00	3194

HORTONWORKS

Materialized view maintenance

HORTONWORKS

Rebuilding materialized views

- Rebuild needs to be triggered manually by user

```
ALTER MATERIALIZED VIEW [db_name.]materialized_view_name REBUILD;
```

- **Incremental** materialized view maintenance
  - Only refresh data that has changed in source tables
  - Multiple benefits
    - Decrease rebuild step execution time
    - Preserves LLAP cache for existing data
  - Materialized view should only use transactional tables (micromanaged or ACID)
  - Current implementation only supports incremental rebuild for insert operations
    - Update/delete operations force full rebuild
- Optimizer will attempt incremental rebuild
  - Otherwise, fallback to **full rebuild** (`INSERT OVERWRITE` with MV definition)

HORTONWORKS

Incremental view maintenance algorithm

- Relies on materialized view rewriting algorithm
  - Materialized view stores write ID for its tables when it is created/refreshed
    - Write ID associates rows with transactions
  - When rebuild is triggered, introduce filter condition on write ID column in MV definition
    - Read only new rows from source tables
  - Execute materialized view rewriting
  - Rewrite `INSERT OVERWRITE` (full rebuild) into more efficient plan
    - `INSERT` (table scan, filter, project, join)
    - `MERGE` (table scan, filter, project, join, aggregate)

HORTONWORKS

Incremental view maintenance algorithm (example)

```
CREATE MATERIALIZED VIEW mv1 AS
SELECT page, user,
    SUM(added) AS c_added,
    SUM(removed) AS c_rmv
FROM wiki
GROUP BY page, user;
```

New records

wiki contents

page	user	…	added	removed	…	writeID
…	…	…	…	…	…	…
Miley	Ashu	…	68	16	…	10000
Justin	Zaka	…	392	239	…	10000

⇢ `ALTER MATERIALIZED VIEW mv1 REBUILD;`

mv1 contents

page	user	c_added	c_rmv
Justin	Boxer	1800	25
Justin	Reach	2912	42
Ke$ha	Xeno	1953	17
Ke$ha	Helz	3194	170
Miley	Ashu	2232	34

HORTONWORKS

Incremental view maintenance algorithm (example)

```
CREATE MATERIALIZED VIEW mv1 AS
SELECT page, user,
    SUM(added) AS c_added,
    SUM(removed) AS c_rmv
FROM wiki
GROUP BY page, user;
```

wiki contents

page	user	…	added	removed	…	writeID
…	…	…	…	…	…	…
Miley	Ashu	…	68	16	…	10000
Justin	Zaka	…	392	239	…	10000

New records

① Rebuild statement rewriting

```
INSERT OVERWRITE mv1
SELECT page, user, SUM(added) AS c_added, SUM(removed) AS c_rmv
FROM (
  SELECT page, user, c_added, c_removed
  FROM mv1
  UNION ALL
  SELECT page, user, SUM(added) AS c_added, SUM(removed) AS c_rmv
  FROM wiki
  WHERE writeID > 9999
  GROUP BY page, user) subq
GROUP BY page, user;
```

Rollup data

mv1 contents

page	user	c_added	c_rmv
Justin	Boxer	1800	25
Justin	Reach	2912	42
Ke$ha	Xeno	1953	17
Ke$ha	Helz	3194	170
Miley	Ashu	2232	34

HORTONWORKS

Incremental view maintenance algorithm (example)

```
CREATE MATERIALIZED VIEW mv1 AS
SELECT page, user,
    SUM(added) AS c_added,
    SUM(removed) AS c_rmv
FROM wiki
GROUP BY page, user;
```

New records

wiki contents

page	user	…	added	removed	…	writeID
…	…	…	…	…	…	…
Miley	Ashu	…	68	16	…	10000
Justin	Zaka	…	392	239	…	10000

② Rewrite `INSERT OVERWRITE` into `MERGE` statement

```
MERGE INTO mv1
USING (
  SELECT page, user, SUM(added) AS c_added, SUM(removed) AS c_rmv
  FROM wiki
  WHERE writeID > 9999
  GROUP BY page, user) src
ON mv1.page = src.page AND mv1.user = src.user
WHEN MATCHED
    THEN UPDATE SET c_added = mv1.c_added + src.c_added,
                    c_removed = mv1.c_removed + src.c_rmv
WHEN NOT MATCHED
    THEN INSERT VALUES (page, user, c_added, c_rmv);
```

mv1 contents

page	user	c_added	c_rmv
Justin	Boxer	1800	25
Justin	Reach	2912	42
Ke$ha	Xeno	1953	17
Ke$ha	Helz	3194	170
Miley	Ashu	**2300**	**50**
Justin	Zaka	**392**	**239**

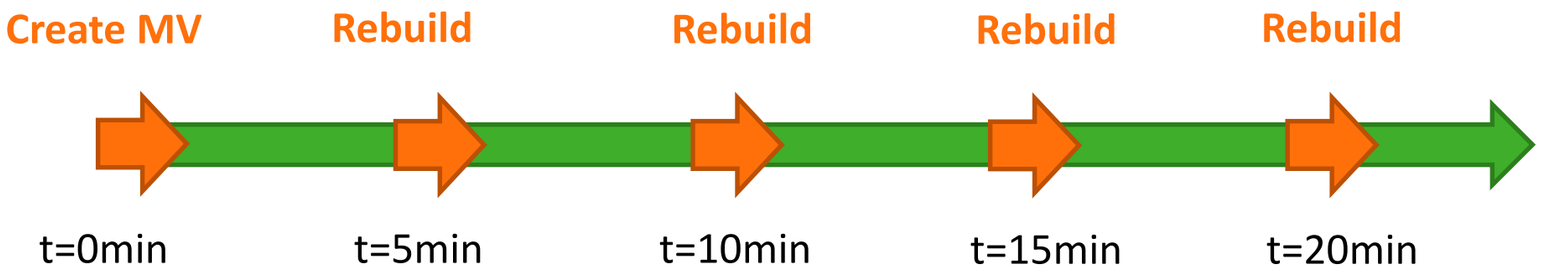
HORTONWORKS

Materialized view lifecycle

HORTONWORKS

Management of materialized view lifecycle

- Do not accept stale data (default)
  - If content of the materialized view is not fresh, we do not use it for automatic query rewriting
  - Still possible to trigger partial rewritings that read both the stale materialized view and new data from source tables
- Accept stale data
  - *Freshness* defined as a time parameter
    - If MV was not rebuilt for a certain time period and there were changes in base tables, ignore
  - `SET hive.materializedview.rewriting.time.window=10min;`
    - Can also be overriden by a certain materialized view using table properties
  - Periodically rebuild materialized view, e.g., every 5 minutes



HORTONWORKS

Road ahead

HORTONWORKS

Road ahead

- Improvements to current materialized views implementation
  - Rewriting performance and scalability
    - Single/many MVs
  - Control physical distribution of data
    - `PARTITIONED BY`, `DISTRIBUTE BY`, `SORT BY`, `CLUSTER BY`
  - Increase incremental view maintenance coverage
    - Support update/delete in source tables

- Materialized view recommender
  - Ease the identification of access patterns for a given workload

HORTONWORKS

Thank you

https://cwiki.apache.org/confluence/display/Hive/Materialized+views