Apache Hive for Oracle DBAs

Luís Marques

REDGLUE

About me

Oracle ACE Alumnus

Long time open source supporter

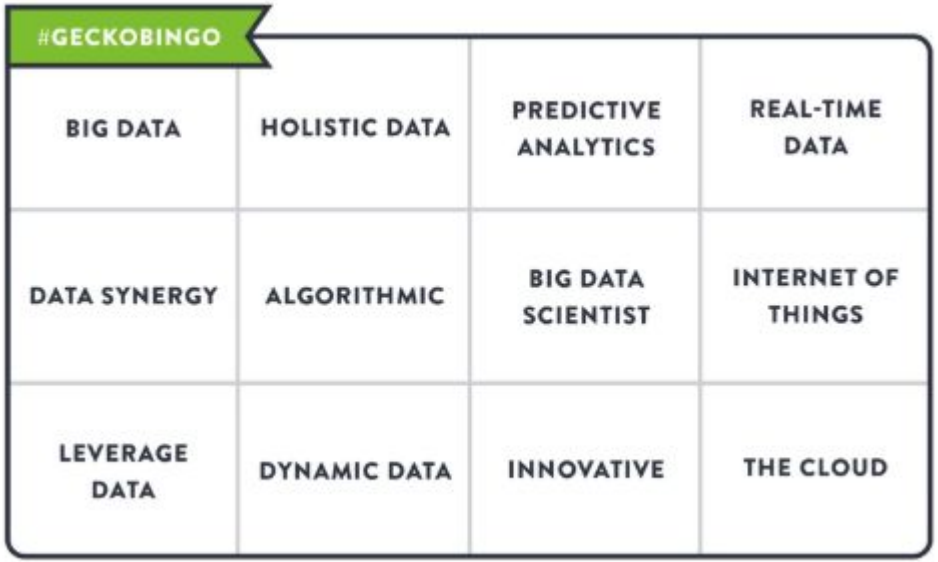
Founder of Redglue

(www.redglue.eu)

works for **@redgluept** as **Lead Data Architect**

@drune

After this talk, you can “buzzword”



The reason behind Hive

Instead of complex MR jobs

```
class Mapper
   method Map(null, tuple [join_key k, value v1, value v2,...])
      Emit(join_key k, tagged_tuple [set_name tag, values [v1, v2, ...] ] )

class Reducer
   method Reduce(join_key k, tagged_tuples [t1, t2,...])
      H = new AssociativeArray : set_name -> values
      for all tagged_tuple t in [t1, t2,...]     // separate values into 2 arrays
         H{t.tag}.add(t.values)
      for all values r in H{'R'}                 // produce a cross-join of the two arrays
         for all values l in H{'L'}
            Emit(null, [k r l] )
```

You have declarative language...

```
SELECT Region, count(*) AS count
FROM item
WHERE Region is not null
GROUP BY Region
```

What is Apache Hive?

- open source, TB/PB scale date warehousing framework based on Hadoop
- The first and more complete **SQL-on-”Hadoop”**
- Revisions SQL:2003 and SQL:2011 compatible
- Data store on several formats
- Several execution engines available
- Interactive Query support (In-memory cache)
- ACID Support

What is Apache Hive?

- Datawarehouse/OLAP activities (data mining, data exploration, batch processing, ETL, etc) - “The heavy lifting of data”
- Low cost scaling, built as extensibility in mind
- Use large datasets (gigabytes/terabytes) scale
- ACID exists (be careful)

Data Model (data units & types)

- Supports primitive column types (integers, numbers, strings, date time and booleans)
- Supports complex types: *Structs*, *Maps* and *Arrays*
- Concept of **databases**, **tables**, **partitions** and **buckets**
- **SerDe**: **ser**ialize and **de**serialized - Allows Hive to read any table and write any *custom* format
- Built-in SerDe: Avro, Parquet, ORC, CSV, etc

Data Model (partitions & bucketing)

- **Partitioning:** used for distributing load horizontally,performance benefit, organization data

```
PARTITIONED BY (flightName STRING, AircraftName STRING)
/employees/flightName=ABC/AircraftName=XYZ
```

- **Buckets (clusters)**: cluster datasets into more manageable parts. Each bucket is a file.

The value of flighID will be hashed by a user-defined number into buckets. The records with the same hash value of flightID will always be stored in the same bucket (same segment of file). This allows bucket-by-bucket joins.

```
CLUSTERED BY (flightID) INTO 256 BUCKETS;
```

HiveQL

- HiveQL is the “SQL” from Hive, like PL/SQL

- Supports **DDL** and **DML** *(as you know it)*
- Supports multi-table inserts

- Supports UDF, UDAF UDTF
  - *`public class Strip extends UDF { … }`*
  - hive> CREATE TEMPORARY FUNCTION STRIP AS 'org.hardik.letsdobigdata.Strip';
  - hive> **select strip('hadoop','ha') from dummy;**

DDL (some examples)

HIVE> CREATE DATABASE/SCHEMA, TABLE, VIEW, INDEX

HIVE> DROP DATABASE/SCHEMA, TABLE, VIEW, INDEX

HIVE> TRUNCATE TABLE

HIVE> ALTER DATABASE/SCHEMA, TABLE, VIEW

HIVE> SHOW DATABASES/SCHEMAS, TABLES, TBLPROPERTIES, VIEWS, PARTITIONS, FUNCTIONS

HIVE> DESCRIBE DATABASE/SCHEMA, table_name, view_name

File formats

- **What Hive can read and write (Parquet, ORC, Avro, GZip, LZO, XML, JSON, your own format, etc…)**

- **Parquet**: compressed, efficient columnar data representation available to any project in the Hadoop
- **ORC**: made for Hive, support Hive type model,columnar storage, block compression, predicate pushdown, ACID*, etc
- Compressed file formats (LZO, *.GZIP)

ORC

- *Stored as columns and compressed = smaller disk reads*
- ***ORC** has a built-in index, min/max values, and other aggregates (eg: sum,max) = skip entire blocks to speed up reads*
- ***ORC** implements predicate pushdown and bloom filters*
- ***ORC** scale*
- *You should use it :-)*

ORC (almost) Internals

- Break file in sets of rows called **stripes**
  - 256M stripes (default)
  - Efficient reading size for large scans on HDFS
  - **Stripes structure**: **data**, **indexes**, **footer**
- **Footer** of the file:
  - Count, Min, Max, sum of all columns (remember Exadata Storage indexes?)
  - List of stripes locations

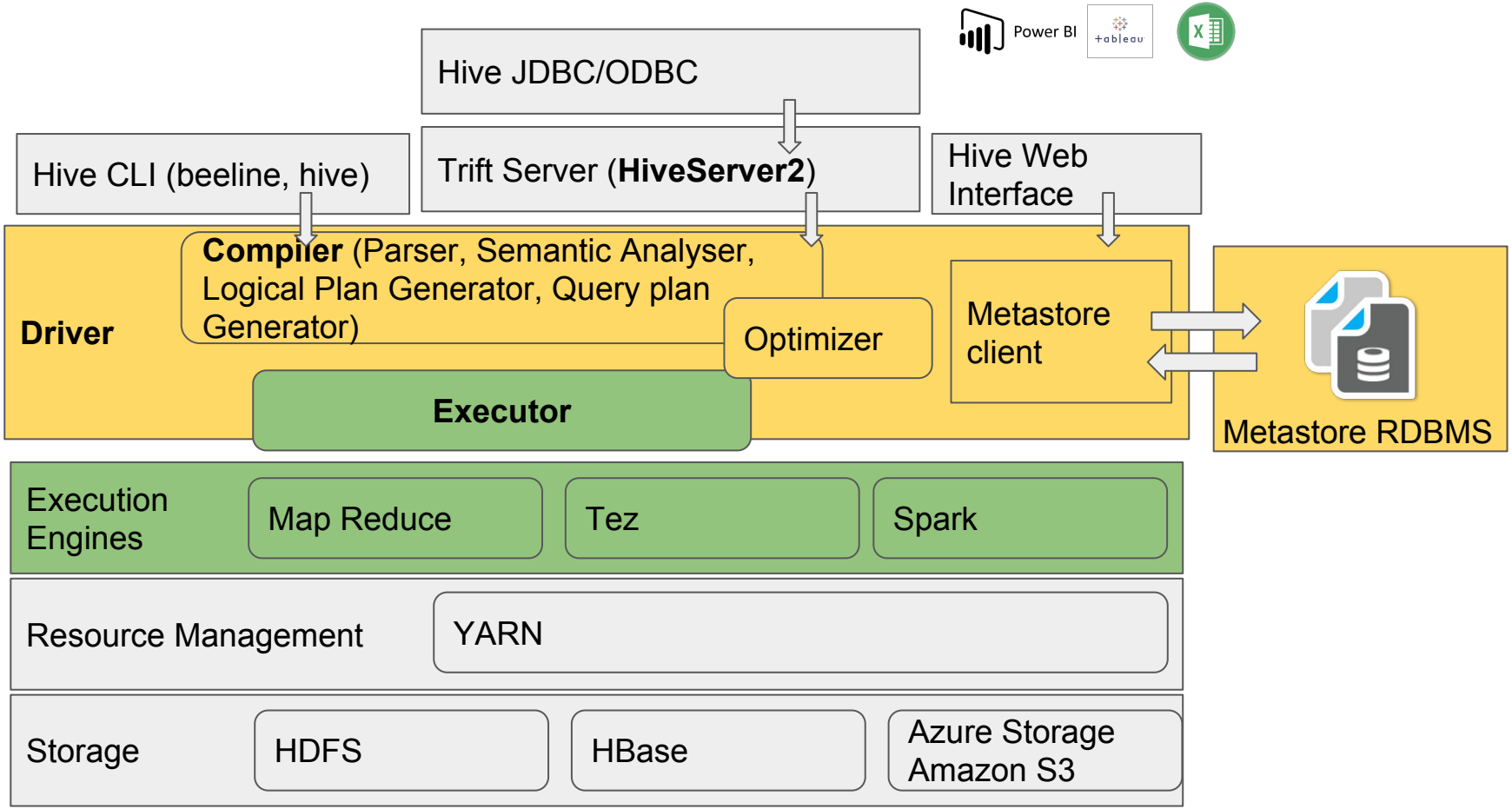
- **PostScript** of the file:
  - Compression parameters (type: none, Snappy, LZO, Zlib)

Indexing - Do we need it?

- **Quick answer: NO**
- **Deprecated in favor of built-in index of columnar formats built-in indexes (ORC, PARQUET)**
- ORC has build in Indexes which allow the format to skip blocks of data during read
- Indexes in Hive **are not like indexes in other databases**.

Some sort of Demo

Hive Architecture

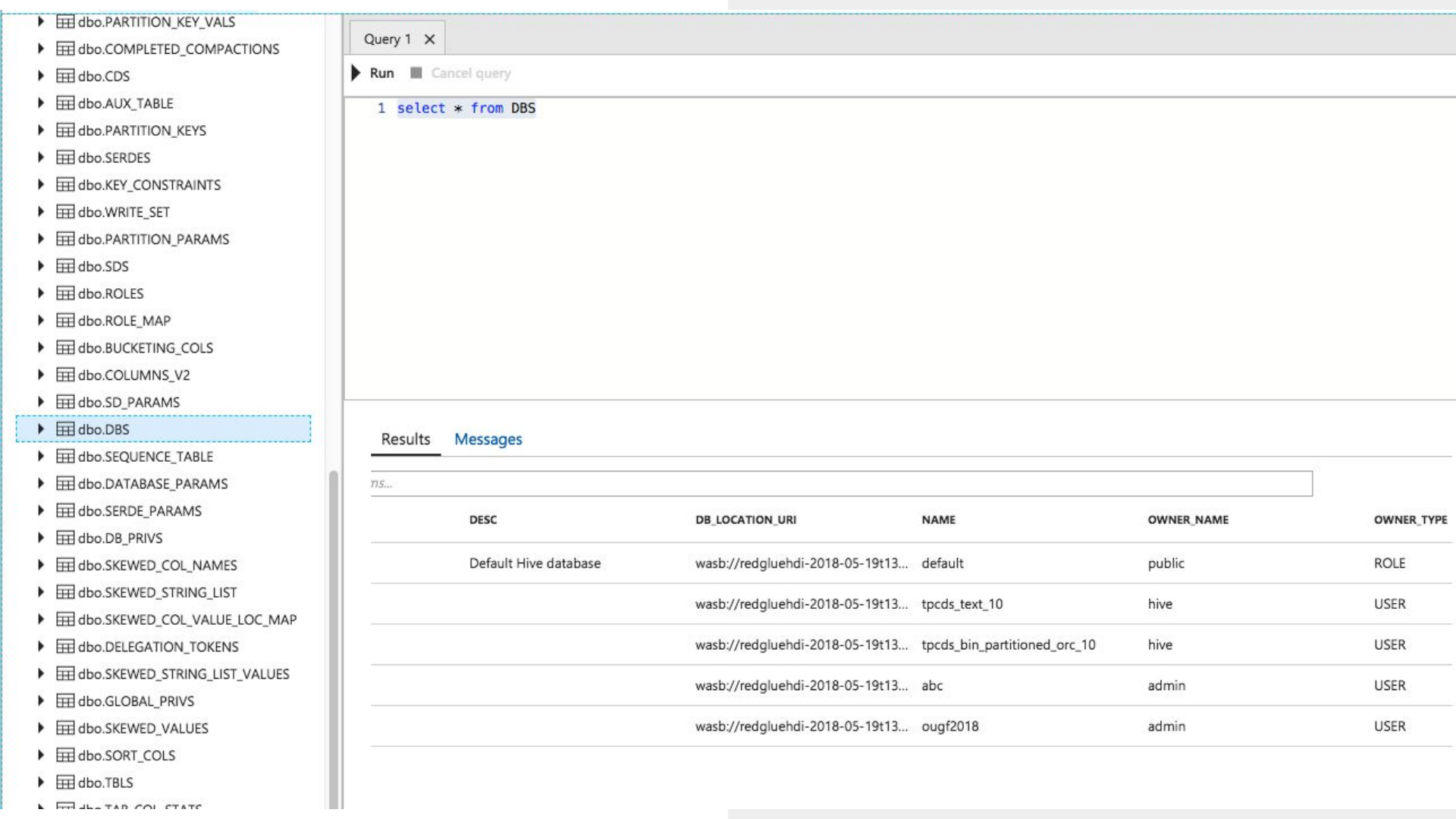


Metastore

- Typically stored in a **RDBMS** (MySQL; SQLServer; PostgreSQL, Derby*) - ACID and concurrency on metadata querys
- **Contains**: metadata for databases, tables or partitions

- **Data abstraction:** provide information about data formats, extractors and loaders in table creation and reused, (ex: dictionary tables - Oracle)

- **Data discovery:** discover relevant and specific data, allow other tools to use metadata to explore data (Ex: SparkSQL)


dbo.PARTITION_KEY_VALS
dbo.COMPLETED_COMPACTIONS
dbo.CDS
dbo.AUX_TABLE
dbo.PARTITION_KEYS
dbo.SERDES
dbo.KEY_CONSTRAINTS
dbo.WRITE_SET
dbo.PARTITION_PARAMS
dbo.SDS
dbo.ROLES
dbo.ROLE_MAP
dbo.BUCKETING_COLS
dbo.COLUMNS_V2
dbo.SD_PARAMS
dbo.DBS
dbo.SEQUENCE_TABLE
dbo.DATABASE_PARAMS
dbo.SERDE_PARAMS
dbo.DB_PRIVS
dbo.SKEWED_COL_NAMES
dbo.SKEWED_STRING_LIST
dbo.SKEWED_COL_VALUE_LOC_MAP
dbo.DELEGATION_TOKENS
dbo.SKEWED_STRING_LIST_VALUES
dbo.GLOBAL_PRIVS
dbo.SKEWED_VALUES
dbo.SORT_COLS
dbo.TBLS
Query 1
Run
Cancel query
1 select * from DBS
Results
Messages
DESC
DB_LOCATION_URI
NAME
OWNER_NAME
OWNER_TYPE
Default Hive database
wasb://redgluehdi-2018-05-19t13...
default
public
ROLE
wasb://redgluehdi-2018-05-19t13...
tpcds_text_10
hive
USER
wasb://redgluehdi-2018-05-19t13...
tpcds_bin_partitioned_orc_10
hive
USER
wasb://redgluehdi-2018-05-19t13...
abc
admin
USER
wasb://redgluehdi-2018-05-19t13...
ougf2018
admin
USER

Execution engines

- 3 execution engines are available:
  - MapReduce (mr)
  - **Tez**
  - Spark

MR: The original, most stable and more reliable, batch oriented, disk-based parallel (like traditional Hadoop MR jobs).

Tez: High performance batch and interactive data processing. Stable in 95% of the time. The one that you should use. Default on HDP.

Spark: Uses Apache Spark (*in-memory* computing platform), High-performance (like Tez), good progress on stability

MapReduce vs Tez/Spark

MapReduce:

- One pair of *map* and *reduce* does one level of aggregation over the data. Complex computations typically require multiple such steps.

Tez/Spark:

- **DAG (Directed Acyclic Graph)**
- The graph **does not have cycles** because the fault tolerance mechanism used by Tez is re-execution of failed tasks
- The limitations of MapReduce in Hadoop became a key point to introduce DAG
- **Pipelining consecutive map steps into one**
- Enforce concurrency and serialization between MapReduce jobs

Tez & DAGs

Two main components:

- **vertices** - in the graph representing processing of data
  - user logic, that analyses and modifies the data, sits in the vertices

- **edges** - representing movement of data between the processing
  - Defines routing of data between tasks (One-To-One, Broadcast Scatter-Gather)
  - Defines when a consumer task is scheduled (Sequential, Concurrent)
  - Defines the lifetime/reliability of a task output

Hive Cost Based Optimizer - Why

- Distributed SQL query processing in Hadoop differs from conventional relational query engine when it comes to handling of intermediate result sets
- Query processing requires sorting and reassembling of intermediate result set - *shuffling*
- Most of the existing optimizations in Hive are about **minimizing shuffling cost** and logical optimizations like filter ***predicate push down***, ***column projection*** **and** ***partition pruning***
- Join reordering and join algorithm possible with cost based optimizer.

Hive CBO - What to get

- Based on a project called Apache Calcite (https://calcite.apache.org/)

- You can get using a **Cost Based Optimizer**:
  - How to order Join (join reordering)
  - Algorithm to use for a Join
  - Intermediate result be persisted or should it be recomputed on failure
  - degree of parallelism at any operator (number of mappers and reducers
  - Semi Join selection
  - (others optimizer tricks like histograms)

ACID: Implementation

- Adding data to Hive was only possible **adding partitions** or new **tables**
- **Update** or **delete** data required **delete partition** and add it again

- Problem?
  - *Data is changing..all the time*

- **ACID** was needed (not OLTP ACID - millions of transactions per hour) - Support of millions of records per transaction

- Support for Flume, Storm or Spark streaming to write directly to Hive tables in a transaction and “SCD”

ACID: Implementation

- **Compaction** - The key for Hive ACID implementation
  - Modifications are stored in deltas:
    - **Major compaction** on 10% of table modified (in delta)
    - **Minor compaction** on 10 deltas
  - Metastore will schedule the Compaction process
  - Only suitable for batch processing
  - Support only for ORC file format

LLAP - The in-memory engine

- More daemons - LLAP daemons run on top of YARN
- In-memory caching layer with async I/O
- Priority execution and fast concurrent execution
- LLAP brings compute to memory (rather than compute to disk)
- Works on top of Tez and Spark
- It looks easy to setup on Amari (the reality is that a lot of tunning is required)

Some sort of Demo

Oracle and Hive integration

dbms_hadoop.create_extddl_for_hive(
CLUSTER_ID=>'dw-hadoop-cluster', DB_NAME=>'DW',
HIVE_TABLE_NAME=>'transac', HIVE_PARTITION=>FALSE,
TABLE_NAME=>'transac_ext', PERFORM_DDL=>FALSE,
TEXT_OF_DDL=>**DDLtxt**);

```
CREATE TABLE transac_ext (  order_id NUMBER,  order_date DATE,  ... )  ORGANIZATION
EXTERNAL
(TYPE
ORACLE_HIVE
DEFAULT DIRECTORY DEFAULT_DIR
ACCESS PARAMETERS
(com.oracle.bigdata.cluster=dw-hadoop-cluster
com.oracle.bigdata.tablename=DW.transac))
```

LLAP - Real Customer Data

Query	Period	Oracle RAC	1º Execut	2º Execut	3º Execution
Query 1	5 months	4m	10m30s	9min	5m16s
Query 2	5 months	8m	10m29s	9min	5m16s
Query 3	1 Year (2016)	3 hours	22m39s	9m50s	
Query 4	1 Year (2016)	4 hours	9m34s	9m46s	
Query 5	17 months	Don't run (more	17m40s	9m52s	
Query 2 + 3 + 4		6 hours	27m08s	17m20s	

Thank you

Questions?

@drune
https://www.linkedin.com/in/lcmarques/

luis.marques@redglue.eu

@redgluept
www.redglue.eu

REDGLUE