

Apunts CE_5075 4.1

lloc: [Institut d'Ensenyaments a Distància de les Illes
Balears](#)

Curs: Big data aplicat

Llibre: Apunts CE_5075 4.1

Imprès per: Carlos Sanchez Recio

Data: dimarts, 24 de desembre 2024, 14:59

Taula de continguts

1. Introducció

1.1. Característiques principals d'Impala

2. Arquitectura

2.1. Emmagatzematge

2.2. Gestió de Metadades

3. SQL en Impala

4. Impala en la màquina virtual Cloudera QuickStart

5. Tipus de dades complexos

5.1. Un primer exemple

5.2. Un segon exemple

5.3. Càrrega de fitxers JSON (només per a Hive)

5.4. Càrrega de fitxers JSON (per a Hive i Impala)

6. Particionament

7. Impala i HBase

8. Bibliografia

1. Introducció

Apache Impala és un **motor d'execució de consultes SQL d'alt rendiment** dissenyat per a treballar amb grans volums de dades emmagatzemades en sistemes Hadoop, ja sigui directament en HDFS, com en Apache HBase i Apache Kudu, dues bases de dades columnars de l'ecosistema Hadoop. Així mateix, Impala també suporta altres sistemes d'emmagatzematge del nivell com Amazon S3 o Azure Data Lake Store.

Apache Impala va ser desenvolupat per Cloudera, l'empresa líder en solucions per a big data durant molts anys, i es va llançar com a projecte de codi obert el 2013 sota la llicència Apache 2.0. Poc després, Impala va passar a ser un projecte Apache, cosa que va permetre la seva adopció ràpida per part de la comunitat.

Tot i que Apache Hive també ofereix capacitats SQL en aquest entorn, Impala va sorgir per abordar una limitació clau de Hive: **el rendiment en temps real de les consultes interactives**.

Ja sabem que Hive es va dissenyar com una eina que permetés als analistes escriure consultes SQL sobre dades emmagatzemades a Hadoop (HDFS i HBase), executant aquestes consultes mitjançant treballs MapReduce. Aquest enfocament va ser revolucionari, però també tenia limitacions importants:

- **Lentitud en l'execució de consultes:** les consultes SQL de Hive sovint triguen molt de temps a executar-se, ja que MapReduce no està optimitzat per a tasques interactives o d'anàlisi en temps real.
- **Escenaris inadequats per a interactivitat:** Hive és excel·lent per a càrregues de treball *batch* i informes programats, però resulta poc eficient quan necessitem analitzar dades de forma dinàmica o immediata.
- **Ús intensiu de recursos:** l'arquitectura basada en MapReduce pot requerir molts recursos fins i tot per a operacions senzilles, fet que en limita l'eficàcia en entorns on la latència és crítica.

Impala, va aparèixer en 2013 com una resposta a aquestes limitacions, amb un objectiu clar: **oferir consultes SQL interactives i d'alt rendiment sobre Hadoop, mantenint la compatibilitat amb l'ecosistema existent**. Les seves característiques clau inclouen:

- **Execució en memòria:** Impala evita l'ús de MapReduce i executa les consultes directament en memòria, reduint significativament la latència.
- **Temps de resposta quasi instantani:** dissenyat per a analistes i aplicacions interactives, Impala permet explorar grans conjunts de dades en temps real, cosa que no és possible amb Hive.
- **Integració completa amb Hadoop:** igual que Hive, Impala treballa amb HDFS, Apache HBase i el Hive Metastore, assegurant que les dades i esquemes siguin consistents entre ambdós sistemes.

En tot cas, tot i que Hive i Impala ofereixen funcionalitats SQL, no són eines exclusivament alternatives, sinó complementàries. Cada eina està optimitzada per a diferents casos d'ús:

- **Hive:** per a processament *batch*, tasques ETL i càrregues de treball programades.
- **Impala:** per a consultes interactives i anàlisi ad hoc amb una latència mínima.

Un aspecte important a destacar és que **Impala pot emprar el magatzem de metadades de Hive**, cosa que permet la compatibilitat amb les estructures d'emmagatzematge existents. Això significa que tant Hive com Impala poden utilitzar les mateixes taules, vistes i metadades, assegurant que els esquemes siguin consistents entre ambdues eines. Això no només facilita la migració entre eines, sinó que també permet als usuaris explotar les dades d'una manera més integrada i interoperable, aprofitant les característiques de cadascuna per als casos d'ús més adequats.

Des de la seva aparició, Impala ha estat adoptat àmpliament gràcies a les seves capacitats per processar grans volums de dades amb velocitats comparables a les bases de dades tradicionals. No obstant això, amb l'auge d'Apache Spark (el veurem en detall en els lliuraments 7 i 8), que ofereix una funcionalitat més àmplia per a l'anàlisi de dades en temps real i de big data, alguns casos d'ús d'Impala han estat substituïts per Spark, que

combina la velocitat de l'execució en memòria amb una compatibilitat més gran amb altres formats d'emmagatzematge i mètodes d'anàlisi. En tot cas, el projecte d'Impala continua evolucionant, amb una comunitat activa que contribueix a millorar-ne el rendiment, la compatibilitat i la seguretat.



En aquest curs estam veient l'evolució dels sistemes distribuïts de big data.

La primera aproximació va ser programar els treballs **MapReduce** en **Java**. Posteriorment, apareix **Pig**, que amb el seu llenguatge Pig Latin, ofereix un major nivell d'abstracció i facilita molt l'escriptura dels treballs MapReduce. Després hem vist **Hive**, que una vegada que hem definit un magatzem de dades, ens permet interactuar amb les dades emprant **SQL**, independentment de com estiguin emmagatzemades. **Impala** és la següent passa, on també interactuam mitjançant SQL, però d'una manera molt més eficient, sense utilitzar treballs MapReduce. La darrera passa en aquesta evolució és **Spark**, també molt eficient, que pot treballar tant sobre HDFS com amb altres formats d'emmagatzematge, destacant en entorns *cloud*. Spark proporciona, a més, llibreries molt potents per a l'anàlisi de dades i l'aprenentatge automàtic amb dades massives.

IMPORTANT

1.1. Característiques principals d'Impala

Veiem a continuació un resum de les principals característiques d'Apache Impala.

1. Execució en memòria

Impala evita l'ús de MapReduce, utilitzant un motor d'execució optimitzat que processa les dades directament a la memòria RAM. Això redueix significativament la latència, fent-lo adequat per a consultes interactives.

2. Arquitectura distribuïda

Impala funciona en un entorn distribuït, on múltiples nodes col·laboren per processar consultes en paral·lel, aprofitant al màxim els recursos del clúster.

3. Compatibilitat amb formats de dades avançats

Suporta formats columnars com Parquet i ORC, ideals per a anàlisi de dades optimitzada. També permet fer feina amb arxius de text (CSV o JSON), tot i que de manera molt menys eficient. També suporta, amb algunes limitacions AVRO.

4. Integració amb el Hive Metastore

Impala comparteix la informació d'esquemes amb Apache Hive, assegurant que les taules i bases de dades es puguin utilitzar indistintament entre ambdues eines.

5. Optimització per a la interactivitat

Les seves consultes són ràpides, i inclou suport per funcions SQL avançades com subconsultes, funcions analítiques i finestres, ideals per a exploració de dades i visualització.

6. Integració amb eines d'empresa

És compatible amb JDBC i ODBC, permetent la seva connexió amb eines populars de *Business Intelligence* com Power BI (que veurem en detall el lliurament 6 de Sistemes de big data), Tableau i altres.

7. Suport per a altres sistemes d'emmagatzematge

A més d'HDFS, Impala ofereix suport directe per a HBase i per a Kudu. També suporta sistemes d'emmagatzematge *cloud* com Amazon S3 o Azure Data Lake Store.

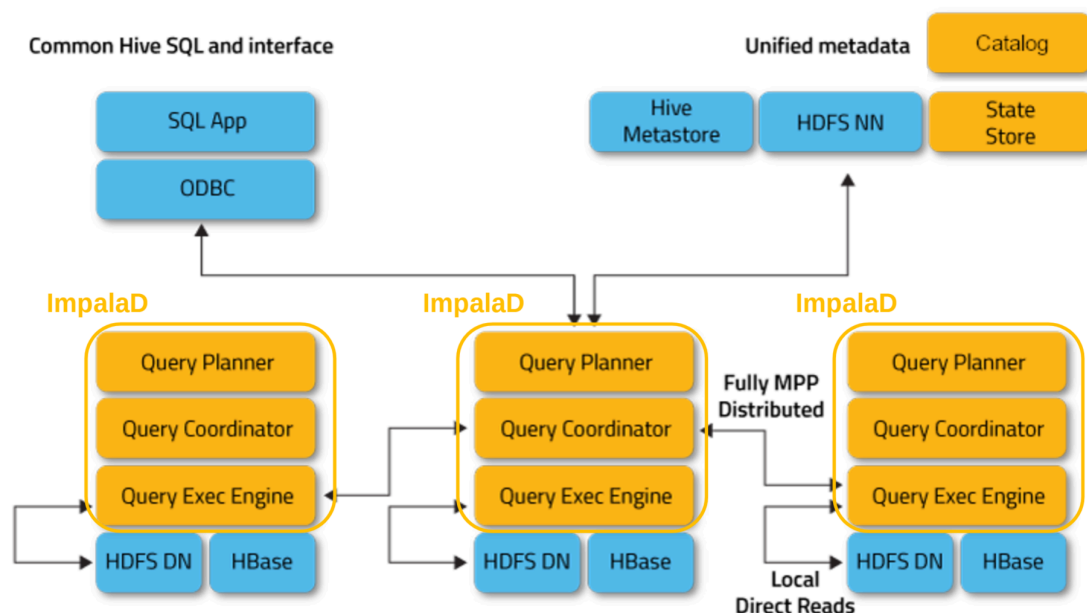
2. Arquitectura

Apache Impala és el que anomenem un **motor d'execució SQL massivament paral·lel** o **MPP** (Massively Parallel Processing), és a dir, una arquitectura per a l'execució de consultes SQL sobre grans volums de dades basada en el paral·lelisme.

Aquesta arquitectura divideix una tasca complexa, com una consulta SQL, en sub-tasques més petites que s'executen en paral·lel en múltiples nodes dins d'un clúster distribuït. Cada node treballa sobre un subconjunt de les dades i, al final, es combina el resultat per oferir una resposta completa i coherent al client.

Com ja sabem, Impala està específicament dissenyat per a executar-se sobre clústers Hadoop. Existeixen altres motors MPP per a entorns *cloud*, com Google BigQuery o Amazon Redshift. Aquests dos combinen les característiques d'un *data warehouse* amb les d'un motor MPP. En canvi, Impala, només té les capacitats d'un motor MPP, no les d'un *data warehouse*, tot i que aquesta funcionalitat es pot aconseguir emprant Hive i el seu *metastore*.

L'arquitectura d'Impala integra components altament especialitzats, que treballen conjuntament per maximitzar el rendiment i assegurar la coherència de les dades. En concret, l'arquitectura d'Impala està formada pels components (*daemons*) que podem veure en aquesta figura (els elements de color groguenc) i que detallam a continuació.



Imatge: Arquitectura d'Apache Impala. Font: <https://impala.apache.org/> (modificada)

Impala Daemon (*impalad*)

És el component clau que s'executa en cada node del clúster i que, a la vegada, conté tres components principals:

- **Query Planner:** Quan un client envia una consulta SQL, el *impalad* divideix la consulta en fragments que s'assignen als nodes segons la localització de les dades. Aquesta planificació cerca minimitzar el moviment de dades.
- **Query Coordinator:** Coordina l'execució de la consulta entre els diversos nodes, recull els resultats parcials, els combina i els retorna al client.
- **Query Executor:** Gestiona l'execució de consultes al node local, llegint directament les dades emmagatzemades en HDFS o Apache HBase en el node local i realitzant els càlculs necessaris.

Aquest enfocament fa que cada node pugui actuar com a coordinador o executor, depenent de les necessitats de la consulta.

Statestore Daemon (*statestored*)

Monitorea l'estat de tots els *impalad* del clúster i assegura que cada node tingui informació actualitzada sobre la disponibilitat dels altres. Encara que s'aturi, el clúster pot continuar operant amb dades emmagatzemades localment.

Catalog Daemon (*catalogd*)

És el responsable de gestionar les metadades necessàries per a l'execució de consultes SQL. Això inclou esquemes de taules, ubicacions de dades, particions i permisos. També s'encarrega de distribuir les metadades (mitjançant actualitzacions incrementals) a tots els nodes del clúster, fent servir el *statestored*. Això permet que les consultes accedeixin a metadades actualitzades en temps real.

2.1. Emmagatzematge

Com ja hem comentat anteriorment, Impala està dissenyat per executar-se sobre clústers Hadoop. Per tant, el més habitual és trobar que les dades estiguin emmagatzemades en HDFS.

Així mateix, Impala pot accedir directament a dades emmagatzemades en HBase, la base de dades NoSQL columnar de l'ecosistema Hadoop, que funciona sobre HDFS. Parlem en detall de HBase en aquest lliurament del mòdul de Sistemes de big data.

Tot i que Apache Kudu, una altra base de dades NoSQL columnar, no formi part de l'ecosistema Hadoop (ja que no s'executa sobre HDFS), també es pot integrar amb Impala. Kudu està dissenyat per manejar càrregues mixtes (lectura i escriptura) amb baixa latència. És ideal per a dades estructurades que requereixen actualitzacions freqüents, com dades d'anàlisis en temps real o registres temporals. Treballa amb un sistema propi d'emmagatzematge distribuït i està ben integrat amb Impala, permetent consultes directes sobre les seves taules.

També ja hem comentat anteriorment que Impala suporta la integració amb sistemes d'emmagatzematge en el nímul:

- **Amazon S3:** Un dels sistemes d'emmagatzematge més comuns per a arquitectures en nímul, que permet accedir a dades mitjançant el model d'objectes, oferint escalabilitat i flexibilitat.
- **Azure Data Lake Store (ADLS) i Azure Blob File System (ABFS):** També són compatibles amb Impala, permetent l'execució de consultes SQL directament sobre dades en l'entorn d'Azure, facilitant la integració amb altres serveis de nímul d'Azure

Per últim, també podem fer feina amb Impala accedint al sistema d'arxius local, cosa que pot ser útil per a proves o entorns molt petits, però que no està pensat per a clústers distribuïts.

Formats de fitxer

Sigui quin sigui el sistema d'emmagatzematge (HDFS, en el nímul o local), hem de tenir present en quins formats poden estar emmagatzemades les dades. Impala té suport natiu de Parquet i ORC, dos formats dissenyats per a l'eficiència en l'emmagatzematge i el processament de dades mitjançant SQL:

- **Parquet:** format columnar comprimit i optimitzat per a l'anàlisi de dades massives.
- **ORC** (Optimized Row Columnar): un altre format columnar, molt utilitzat amb Hive, especialment en contextos de compressió i lectura eficient.

Quan cream una taula hem d'especificar quin format emprarem, mitjançant la clàusula **STORED AS**. Per exemple:

```
CREATE TABLE taula_orc (  
    id INT,  
    nom STRING,  
    edat INT  
)  
STORED AS ORC;
```

Si no s'hi inclou cap **STORED AS**, es farà servir el format per defecte que és Parquet.

És millor emprar Parquet o ORC? Impala té un suport més ampli de Parquet, amb més funcionalitats disponibles, especialment en termes d'optimització i rendiment. En canvi, ORC és el format natiu d'Hive i, per tant, pot ser més adequat per a escenaris on Impala i Hive comparteixen dades.

A més de Parquet i ORC, Impala també pot fer feina amb **Avro** (**STORED AS AVRO**), un format orientat a la serialització de dades, tot i que amb algunes limitacions. La més important és que Impala no suporta l'escriptura de dades en AVRO. A més, el rendiment és pitjor que amb Parquet o ORC.

Per últim, Impala també suporta **arxius de text** (STORED AS TEXTFILE), principalment CSV, emmagatzemats en HDFS. Ens hem d'assegurar d'utilitzar els delimitadors adequats. Podem especificar-los mitjançant la clàusula ROW FORMAT DELIMITED, definint el separador de camps (FIELDS TERMINATED BY), per defecte el caràcter SOH (*Start Of Header*, el caràcter 1 de la taula ASCII, '\001'), i el separador de línies (LINES TERMINATED BY), per defecte el caràcter salt de línia ('\n'). Per exemple, la següent sentència crea una taula a partir d'un arxiu de text emprant el tabulador com a separador de camps i el salt de línia per separar línies.

```
CREATE EXTERNAL TABLE taula_csv (  
  id INT,  
  nom STRING,  
  edat INT  
)  
ROW FORMAT DELIMITED  
FIELDS TERMINATED BY '\t'  
STORED AS TEXTFILE  
LOCATION '/user/cloudera/.../taula';
```

Fixau-vos que aquí hem definit la taula com a externa. Les taules externes funcionen igual que en Hive. Quan cream una taula externa amb CREATE EXTERNAL TABLE, el sistema assumeix que les dades ja existeixen al directori especificat en la clàusula LOCATION i que no ha de controlar-ne la vida útil. En cas contrari, les dades es mouen al directori per defecte d'Impala, que normalment és /user/hive/warehouse/<nom_taula>. En general, les taules externes són útils quan les dades són compartides entre diferents aplicacions i no volem que Impala (o Hive) gestioni l'emmagatzematge o l'eliminació de les dades. Així doncs, quan feim un DROP TABLE d'una taula externa, només s'eliminen les metadades del catàleg, però no les dades en sí.

Independentment de si la taula és externa o no, el tractament de fitxers de text és molt ineficient. És un format basat en files, no en columnes, sense compressió i sense cap optimització per a la distribució de dades. Per aquesta raó, no es recomana emprar-los i és molt habitual, una vegada que tenim una taula com l'anterior *taula_csv*, fer-ne una còpia en format Parquet i treballar amb ella:

```
CREATE TABLE taula_parquet  
STORED AS PARQUET  
AS SELECT * FROM taula_csv;
```

2.2. Gestió de Metadades

La gestió de metadades és una part fonamental del funcionament d'Apache Impala. Les metadades proporcionen informació estructural sobre les taules, els esquemes, les ubicacions i altres atributs necessaris per executar consultes de manera eficient. Aquesta gestió es basa en una combinació del **Metastore de Hive** i els components interns d'Impala.

El Metastore de Hive

Impala utilitza el Metastore de Hive com a repositori centralitzat de metadades. Aquest sistema emmagatzema informació sobre:

- Els esquemes de les taules (columnes, tipus de dades, etc.).
- Les ubicacions dels fitxers de dades a HDFS o altres sistemes de fitxers compatibles.
- Els formats d'emmagatzematge de les dades (PARQUET, AVRO, TEXT, etc.).
- Les particions definides en taules particionades.

El Metastore de Hive és compartit entre Hive i Impala, cosa que facilita la interoperabilitat. Així, qualsevol taula creada o modificada en un dels dos sistemes es pot utilitzar a l'altre.

Components d'Impala relacionats amb les metadades

Impala complementa el Metastore amb els seus propis components que gestionen i optimitzen l'accés a les metadades:

Catalog Server:

- S'encarrega de sincronitzar les metadades entre el Metastore de Hive i els nodes d'Impala.
- Actualitza automàticament les metadades quan es crea o modifica una taula des d'Impala.
- Proporciona informació als nodes d'Impala perquè puguin executar consultes eficientment.

Statestore:

- Coordina la distribució de metadades actualitzades entre els diferents nodes del clúster d'Impala.
- Assegura que tots els nodes d'Impala treballin amb la mateixa visió de les metadades.

Ordres per gestionar les metadades a Impala

A causa de la memòria cau de metadades que utilitza Impala, hi ha casos en què és necessari actualitzar manualment aquestes metadades. Les ordres següents permeten gestionar-les de manera explícita:

INVALIDATE METADATA

Aquesta ordre força Impala a descartar la seva memòria cau i tornar a carregar totes les metadades des del Metastore. S'utilitza aquesta ordre quan es fan canvis al sistema fora d'Impala, com per exemple quan cream o modifiquem una taula des de Hive, o quan afegim fitxers a una ubicació directament a HDFS.

Exemple:

```
INVALIDATE METADATA table_name;
```

REFRESH

Actualitza la memòria cau de metadades per a una taula concreta. És més ràpida que `INVALIDATE METADATA` perquè només sincronitza informació per a la taula especificada, sense descartar totes les metadades. Es recomana utilitzar `REFRESH` quan s'han afegit dades noves als fitxers d'una taula, però l'estructura de la taula no ha canviat.

Exemple:

```
REFRESH table_name;
```

Bones pràctiques per a la gestió de metadades

Compartir el mateix Metastore entre Hive i Impala: per garantir que les taules siguin accessibles des d'ambdós sistemes, és important configurar Hive i Impala per utilitzar el mateix Metastore.

Actualitzar les metadades només quan sigui necessari: l'ús indiscriminat de `INVALIDATE METADATA` pot impactar negativament en el rendiment, ja que implica descartar i tornar a carregar tota la informació de metadades.

Evitar conflictes de versions del Metastore: ens hem d'assegurar que la versió del Metastore sigui compatible amb la versió d'Impala i Hive que s'utilitzen al clúster.

Interoperabilitat i format de taules

Les taules creades amb Impala són compatibles amb Hive si es compleixen les següents condicions:

- S'utilitzen tipus de dades compatibles.
- Els formats d'emmagatzematge són suportats per ambdós sistemes (com `PARQUET`, `AVRO` o `TEXTFILE`).
- Si la taula és externa, les dades han de ser accessibles des de la ubicació especificada (`LOCATION`).

Impacte de les particions en les metadades

Les taules particionades poden generar un gran volum de metadades al Metastore, ja que cada partició s'emmagatzema com un registre separat.

Impala optimitza aquest procés mitjançant l'ús de **lazy loading** (carrega metadades de particions només quan són necessàries).

3. SQL en Impala

Impala proporciona suport per al llenguatge SQL per a l'anàlisi de dades en Hadoop. Aquesta és precisament la capacitat més important d'Impala, ja que permet als usuaris interactuar amb els seus grans conjunts de dades utilitzant un llenguatge conegut i estàndard de la indústria per a consultes de bases de dades. No oblidem que, a més, l'execució de les sentències es fan en un entorn distribuït, de manera paral·lela, amb una baixa latència.

Impala admet una sintaxi SQL estàndard, tant pel que fa a la definició de dades (DDL) com a la seva manipulació (DML) mitjançant insercions, modificacions, esborrats i consultes.

D'altra banda, atès que utilitza el mateix magatzem de metadades que Hive per registrar informació sobre l'estructura i les propietats de les taules, Impala proporciona un alt grau de compatibilitat amb HiveQL, el llenguatge de consultes de Hive. Ja vàrem veure en el lliurament anterior que HiveQL és pràcticament idèntic a SQL. Però té algunes particularitats. Vegem a continuació algunes de les més importants.

Pel que fa al DDL:

- La sentència CREATE TABLE en HiveQL proporciona opcions addicionals. Una d'elles és la que permet definir el format que s'utilitza per emmagatzemar les dades. Per exemple podem especificar "STORED AS PARQUET".
- HiveQL permet particionar les taules per a optimitzar consultes, afegint la clàusula PARTITIONED BY (...) en la sentència CREATE TABLE.
- HiveQL fa servir el tipus de dades STRING en lloc de VARCHAR (tot i que també està suportat en versions recents) i BIGINT en lloc de LONG.
- HiveQL inclou tipus específics per a grans volums de dades, com MAP, ARRAY i STRUCT per treballar amb dades complexes. En parlem amb més detall més endavant.

Pel que fa al DML:

- Quan treballem amb dades massives, en lloc d'emprar ORDER BY, en HiveQL és més habitual emprar SORT BY, que ordena cadascun dels blocs de dades processats pels *mappers* i *reducers*. Això no garanteix un ordre global en els resultats, però és molt més eficient que un ORDER BY, que requereix passar totes les dades per un *reducer*.
- HiveQL proporciona funcions analítiques pròpies, com per exemple PERCENTILE.
- HiveQL incorpora la funció RLIKE (Regular Expression LIKE), que permet realitzar cerques utilitzant expressions regulars dins de cadenes de text. Aquesta funció és especialment útil per analitzar i filtrar dades textuals en taules de Hive. Per exemple: SELECT * FROM log_data WHERE message RLIKE 'ERROR|FATAL';

En l'apartat 5 veurem que Impala amplia la sintaxi de SQL per a donar suport a tipus complexos (ARRAY, MAP i STRUCT) de manera concisa i fàcil d'usar. La nova sintaxi va ser dissenyada per a ser una extensió natural de SQL i proporcionar tota la funcionalitat de SQL amb aquests tipus complexos. Veurem també que aquesta sintaxi que fa servir Impala per a definir consultes amb tipus complexos és diferent de la que proporciona HiveQL.

Abans, però, anem a veure com interactuam amb Impala des de la màquina virtual Cloudera QuickStart que hem estat emprant en lliuraments anteriors.

4. Impala en la màquina virtual Cloudera QuickStart

Impala, com a eina desenvolupada originalment per Cloudera, està disponible en la màquina virtual Cloudera QuickStart que hem estat emprant en el curs.

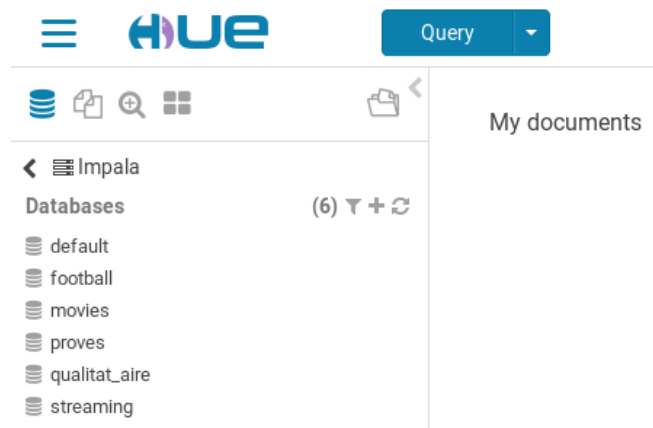
Tenim dues formes principals de treballar amb Impala en la nostra màquina virtual de Cloudera: des de Hue i mitjançant un shell o interfície de línia d'ordres (impala-shell).

Impala des de Hue

La forma més senzilla de fer feina amb Impala és a través de Hue (Hadoop User Experience), la interfície web per consultar bases de dades i magatzems de dades en un entorn Hadoop, que ja vàrem emprar en el lliurament 3 amb Hive.

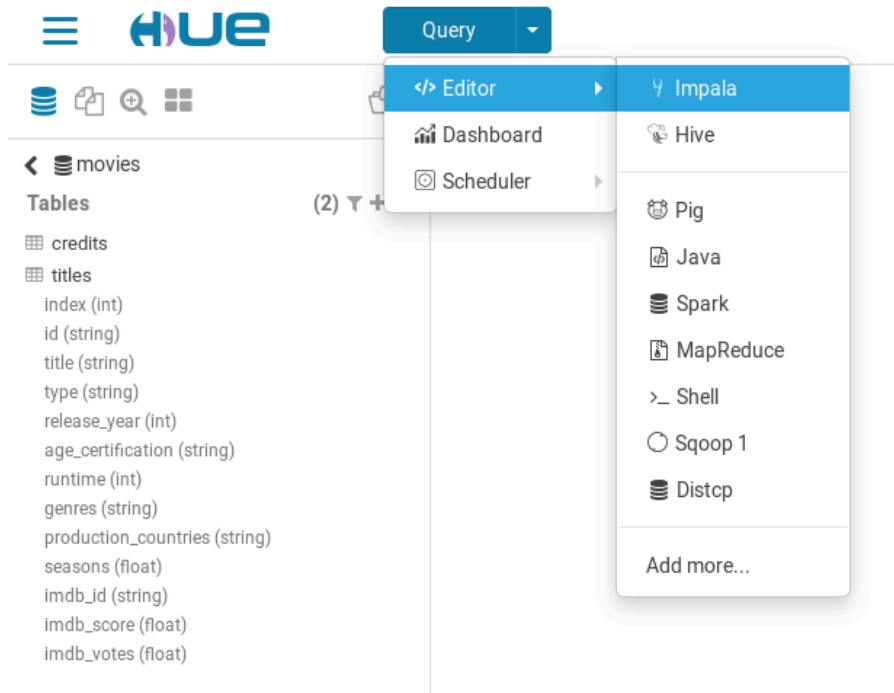
En aquest entorn, Hive i Impala estan integrats, fent servir el mateix metastore, de manera que quan cream una base de dades a Hive, és visible des d'Impala i viceversa.

D'aquesta manera, quan entrem en Hue a les bases de dades d'Impala, podem veure totes les bases de dades que havíem creat amb Hive en el lliurament anterior.



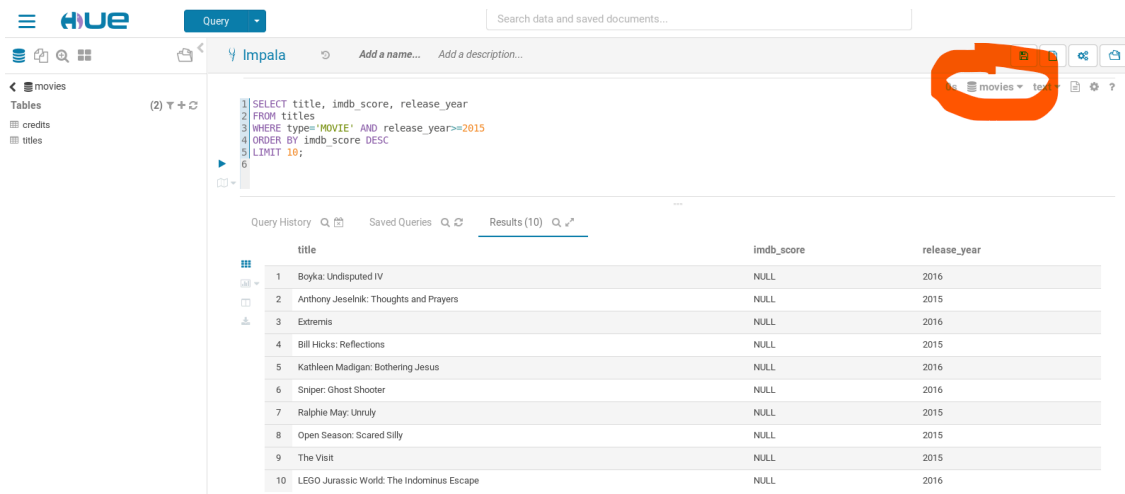
Imatge: Bases de dades Hive disponibles

Anam a treballar amb la base de dades *movies*, amb la qual vàrem fer l'activitat d'aprenentatge, i que conté les taules *credits* i *titles*. En el botó "Query", podem seleccionar l'editor d'Impala per a poder escriure i executar les nostres queries.



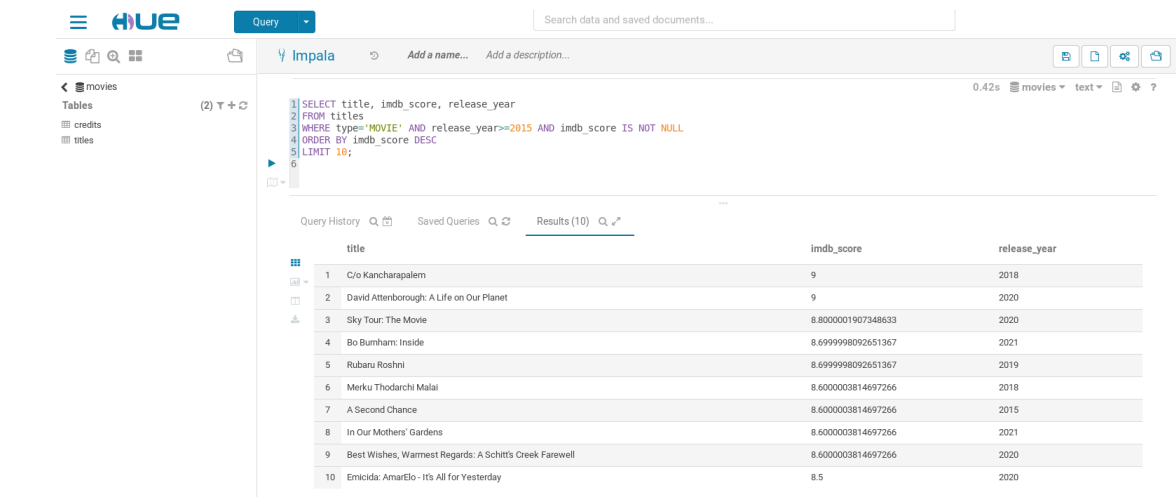
Imatge: Selecció de l'editor d'Impala

Hem de seleccionar la base de dades movies en l'editor d'Impala i escriure i executar la query, en aquest cas, la del subapartat 1.1. Vegem el resultat:



Imatge: Execució d'una query

Sorprenentment, no ens ha donat la mateixa resposta que quan la vàrem executar des del client Hive. El motiu és el tractament dels valors nuls en la columna *imdb_score*, que en Impala els considera com els valors més grans. Per evitar-ho, afegirem " AND *imdb_score* IS NOT NULL " en el WHERE:



```

1 SELECT title, imdb_score, release_year
2 FROM titles
3 WHERE type='MOVIE' AND release_year>2015 AND imdb_score IS NOT NULL
4 ORDER BY imdb_score DESC
5 LIMIT 10;

```

	title	imdb_score	release_year
1	C/o Kanchargapalem	9	2018
2	David Attenborough: A Life on Our Planet	9	2020
3	Sky Tour: The Movie	8.8000001907348633	2020
4	Bo Burnham: Inside	8.69999998092651367	2021
5	Rubaru Roshni	8.69999998092651367	2019
6	Merku Thodarchi Malai	8.6000003814697266	2018
7	A Second Chance	8.6000003814697266	2015
8	In Our Mothers' Gardens	8.6000003814697266	2021
9	Best Wishes, Warmest Regards: A Schitt's Creek Farewell	8.6000003814697266	2020
10	Emicida: AmarElo - It's All for Yesterday	8.5	2020

Imatge: Segona execució de la query

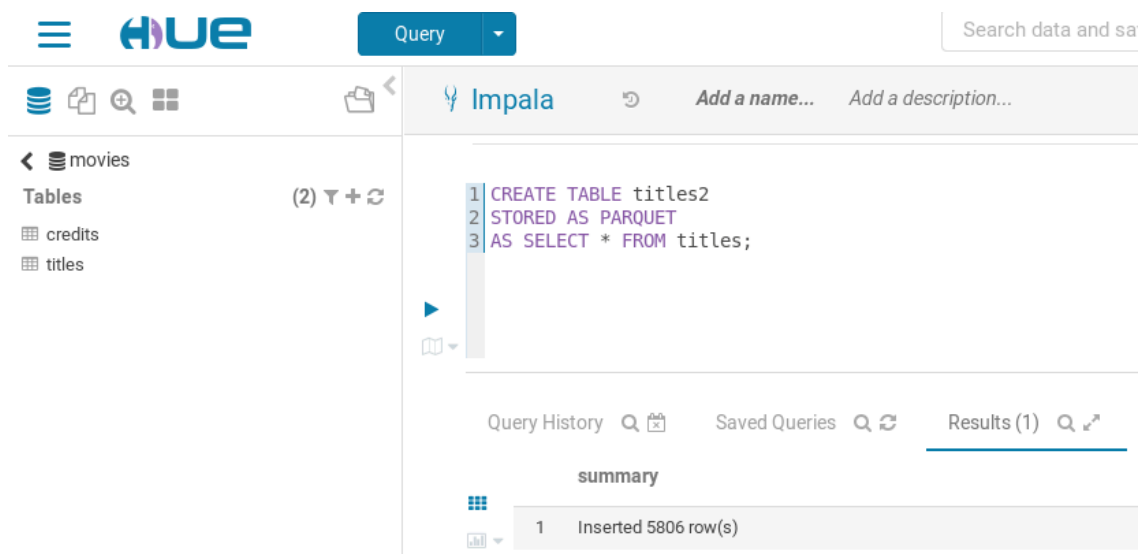
No obstant això, quan varem crear la base de dades movies, ho varem fer directament a partir dels fitxers CSV. Per tant, les dades estan emmagatzemades en format text. Seria convenient fer una còpia de la taula en format Parquet i executar les consultes sobre ella.

Així doncs, executarem la següent sentència:

```

CREATE TABLE titles2
STORED AS PARQUET
AS SELECT * FROM titles;

```



```

1 CREATE TABLE titles2
2 STORED AS PARQUET
3 AS SELECT * FROM titles;

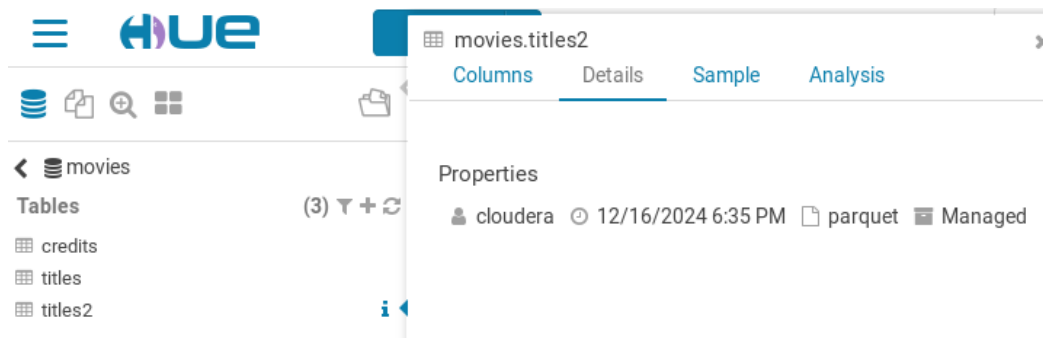
```

summary

1	Inserted 5806 row(s)
---	----------------------

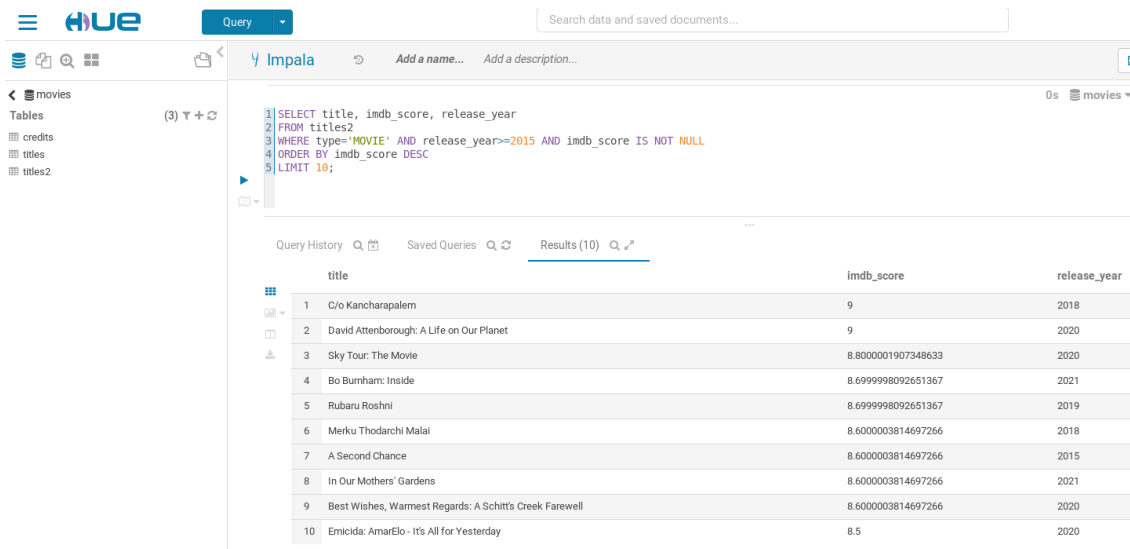
Imatge: Còpia a una nova taula amb format Parquet

Si, a continuació pitjam el botó de recarregar, veurem que ens demana si volem esborrar la *cache*, fer una modificació incremental de les metadades o bé esborrar les metadades i reconstruir l'índex (operació que amb taules distribuïdes en diversos nodes podria ser costosa). Seleccionant qualsevol de les tres podrem veure que ens apareix la nova taula *titles2*. Si miram els detalls de la informació de la taula, podrem comprovar que està emmagatzemada en format Parquet.



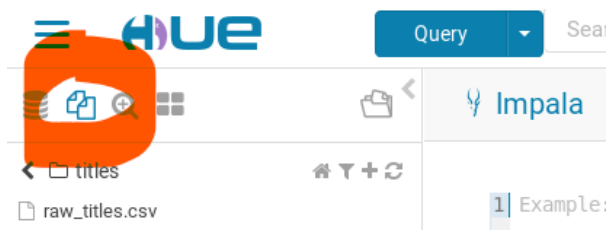
Imatge: Detalls de la nova taula amb format Parquet

I ja podem tornar a executar la nostra query, però ara sobre la taula *titles2*:



Imatge: Execució de la query sobre la taula amb format Parquet

En els exemples anteriors, hem fet feina a partir d'una taula ja creada en Hive, aprofitant que Impala i Hive comparteixen el metastore. Però també podríem haver creat la taula directament des de l'editor d'Impala. Ho farem ara, a partir de l'arxiu [raw_titles.csv](#) que prèviament hem pujat a HDFS al path `/user/cloudera/impala/titles`. Podem comprovar el contingut de HDFS també des de Hue:



Imatge: Contingut del path HDFS `/user/cloudera/impala/titles`

Ara podem crear la taula amb el format TEXTFILE. Hue permet fer la importació a una nova taula de manera gràfica, pitjant la icona de "+" dins de la base de dades. Aquí ho farem amb una sentència SQL CREATE TABLE. Hem d'especificar que fa servir el tabulador com a caràcter de separació i que no ha de tractar la primera línia perquè és una capçalera.


```
CREATE EXTERNAL TABLE titles_csv(
  index INT,
  id STRING,
  title STRING,
  type STRING,
  release_year INT,
  age_certification STRING,
  runtime INT,
  genres STRING,
  production_countries STRING,
  seasons FLOAT,
  imdb_id STRING,
  imdb_score FLOAT,
  imdb_votes FLOAT)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t'
STORED AS TEXTFILE
LOCATION '/user/cloudera/impala/titles/'
TBLPROPERTIES ("skip.header.line.count"="1");
```

Finalment, com ja hem comentat anteriorment, és convenient crear-ne una còpia emmagatzemada en format Parquet:

```
CREATE TABLE titles_parquet
STORED AS PARQUET
AS SELECT * FROM titles_csv;
```

Després d'això, ja podem executar les nostres queries sobre la taula titles_parquet, tal com hem vist abans.

Shell d'Impala

Per obrir el *shell* o interfície de línia d'ordres d'Impala, hem d'executar l'ordre següent:

```
impala-shell
```

```
[cloudera@quickstart ~]$ impala-shell
Starting Impala Shell without Kerberos authentication
Connected to quickstart.cloudera:21000
Server version: impalad version 2.10.0-cdh5.13.0 RELEASE (build 2511805f1eaa991df1460276c7e9f19d819cd4e4)
*****
Welcome to the Impala shell.
(Impala Shell v2.10.0-cdh5.13.0 (2511805) built on Wed Oct 4 10:55:37 PDT 2017)

After running a query, type SUMMARY to see a summary of where time was spent.
*****
[quickstart.cloudera:21000] > █
```

Imatge: Interfície del shell d'Impala

Iniciarem un *shell* interactiu on podem executar sentències SQL. Per exemple, entrarem en la base de dades movies:

```
use movies;
```

I executarem la query que hem vist abans sobre la taula titles_parquet:

```
SELECT title, imdb_score, release_year
FROM titles_parquet
WHERE type='MOVIE' AND release_year>=2015 AND imdb_score IS NOT NULL
ORDER BY imdb_score DESC
LIMIT 10;
```

```
[quickstart.cloudera:21000] > use movies;
Query: use movies
[quickstart.cloudera:21000] > SELECT title, imdb_score, release_year
> FROM titles_parquet
> WHERE type='MOVIE' AND release_year>=2015 AND imdb_score IS NOT NULL
> ORDER BY imdb_score DESC
> LIMIT 10;

Query: select title, imdb_score, release_year
FROM titles_parquet
WHERE type='MOVIE' AND release_year>=2015 AND imdb_score IS NOT NULL
ORDER BY imdb_score DESC
LIMIT 10
Query submitted at: 2024-12-22 11:00:38 (Coordinator: http://quickstart.cloudera:25000)
Query progress can be monitored at: http://quickstart.cloudera:25000/query_plan?query_id=e44417eb393f459f:33ddfd7200000000

+-----+-----+-----+
| title                                     | imdb_score | release_year |
+-----+-----+-----+
| C/o Kancharapalem                       | 9          | 2018         |
| David Attenborough: A Life on Our Planet | 9          | 2020         |
| Sky Tour: The Movie                     | 8.800000190734863 | 2020         |
| Bo Burnham: Inside                      | 8.699999809265137 | 2021         |
| Rubaru Roshni                           | 8.699999809265137 | 2019         |
| Merku Thodarchi Malai                   | 8.600000381469727 | 2018         |
| A Second Chance                         | 8.600000381469727 | 2015         |
| In Our Mothers' Gardens                  | 8.600000381469727 | 2021         |
| Best Wishes, Warmest Regards: A Schitt's Creek Farewell | 8.600000381469727 | 2020         |
| Emicida: AmarElo - It's All for Yesterday | 8.5         | 2020         |
+-----+-----+-----+

Fetched 10 row(s) in 1.68s
```

Imatge: Execució d'una query en el shell d'Impala

5. Tipus de dades complexos

Els tipus complexos (també anomenats tipus niats, *nested types*) permeten representar múltiples valors de dades en una única cella (posició fila-columna) d'una taula. Són diferents dels habituals tipus de columnes com `STRING` o `INT`, anomenats escalars o primitius, que representen un únic valor per a una cella. Impala suporta els tipus complexos `ARRAY`, `MAP` i `STRUCT`, també disponibles en Hive. En canvi, Impala no té suport per al tipus `UNION` que Hive proporciona.

Aquests tipus de dades complexos **ARRAY**, **MAP** i **STRUCT** en Impala són molt útils per emmagatzemar dades amb estructures jeràrquiques o no normalitzades directament dins d'una taula. Aquests tipus permeten representar relacions o llistes sense necessitat de crear taules addicionals, cosa que simplifica l'emmagatzematge i l'anàlisi de dades. Vegem cada un dels tipus complexos suportats per Impala:

- **ARRAY** és una col·lecció ordenada de valors del mateix tipus. Serveix per emmagatzemar múltiples valors dins d'una sola fila. Un exemple del seu ús podria ser emmagatzemar els serveis addicionals que un client ha sol·licitat durant la seva estada (p. ex., esmorzar, spa, etc.).
- **MAP** és una col·lecció de parells clau-valor, on cada clau és única. S'utilitza per emmagatzemar dades associatives. Un exemple del seu ús podria ser guardar l'import detallat per a cada categoria de despesa que ha fet el client (p. ex., {"Esmorzar": 30, "Spa": 50}).
- **STRUCT** és una col·lecció de camps etiquetats, que poden tenir diferents tipus de dades. Permet agrupar múltiples valors relacionats en una sola columna. Un exemple del seu ús podria ser emmagatzemar informació detallada del client, com el nom i l'adreça de correu, en una sola columna.

Aquests tipus de dades complexos no són habituals quan treballem amb bases de dades relacionals o fitxers de text tabulats (CSV), que presenten una estructura de dades rígida. En canvi, sí són especialment útils quan feim feina amb **documents JSON** o bases de dades basades en documents, que suporten una estructura de dades molt més flexible.

En els següents apartats veurem uns exemples per entendre com crear una taula amb tipus complexos, com inserir-hi dades, com fer-hi consultes (tant en HiveQL com en SQL d'Impala) i com treballar amb documents JSON.



Podeu trobar més detalls sobre els tipus complexos en Impala

a https://impala.apache.org/docs/build/html/topics/impala_complex_types.html

També trobareu informació (tot i que sense entrar en gaire detall) sobre els tipus de dades suportats per Hive a <https://cwiki.apache.org/confluence/display/hive/languagemanual+types> i sobre les funcions UDF

(com `NAMED_STRUCT`, `ARRAY` i `MAP`)

a <https://cwiki.apache.org/confluence/display/hive/languagemanual+udf>

5.1. Un primer exemple

En aquest primer exemple, anam a fer feina amb un tipus de dades STRUCT molt senzill, que representa a una persona i que conté dos camps, el nom (de tipus cadena de caràcters) i l'edat (de tipus enter). En el segon exemple, ho complicarem una mica i emprarem també ARRAY i MAP.

Començarem creant una base de dades complex_types en Hive on executarem tots els nostres exemples.

```
CREATE DATABASE complex_types;  
USE complex_types;
```

Allà hi crearem la taula des de Hive, amb la següent sentència:

```
CREATE TABLE persones (  
    persona STRUCT<nom: STRING, edat: INT>  
)  
STORED AS PARQUET;
```

Com podem veure, la taula *persones* té una única columna *persona* que és de tipus STRUCT, que a la vegada conté dos camps: el nom i l'edat. Guardam la taula en format Parquet perquè sigui més eficient.

Ara volem inserir les següents dades:

nom	edat
Joan	50
Aina	30
Pep	40

Per fer-ho, seguint en Hive, emprarem la funció NAMED_STRUCT per definir un valor de tipus STRUCT. Per exemple, per a la nostra primera persona:

```
NAMED_STRUCT(  
    'nom', 'Joan',  
    'edat', 50  
)
```

D'altra banda, Hive no permet fer sentències INSERT INTO ... VALUES ..., així que hem d'emprar un SELECT. Aquestes sentències fan la inserció de les dades de la taula:

```
INSERT INTO persones SELECT NAMED_STRUCT('nom', 'Joan', 'edat', 50) AS persona;  
INSERT INTO persones SELECT NAMED_STRUCT('nom', 'Aina', 'edat', 30) AS persona;  
INSERT INTO persones SELECT NAMED_STRUCT('nom', 'Pep', 'edat', 40) AS persona;
```

Cada inserció és un treball MapReduce diferent, amb la qual cosa l'execució és bastant lenta.

Ara ja podem fer-hi consultes. Seguint en Hive, podem començar recuperant totes les persones:

```
SELECT * FROM persones;
```

Que recupera:

```

+-----+-----+
|      persones.persona      |
+-----+-----+
| {"nom":"Joan","edat":50}    |
| {"nom":"Aina","edat":30}    |
| {"nom":"Pep","edat":40}    |
+-----+-----+

```

Amb el punt (.) podem accedir a cada un dels camps del STRUCT persona. Per exemple, anem a recuperar els noms de les persones amb més de 35 anys:

```
SELECT persona.nom FROM persones where persona.edat>35;
```

Que recupera:

```

+-----+-----+
|  nom  |
+-----+-----+
| Joan  |
| Pep   |
+-----+-----+

```

Consultes amb Impala

Ara anem a veure com treballem amb aquesta taula des d'Impala. La primera cosa que haurem de fer és actualitzar les metadades del catàleg, perquè molt probablement la nostra taula *persones* (ni la base de dades *complex_types*) no serà visible ni des de l'editor Impala de Hue ni des d'impala-shell. Per fer-ho, hem d'executar la sentència:

```
INVALIDATE METADATA complex_types.persones;
```

Ara ja podem seleccionar la base de dades abans de fer-hi consultes.

```
USE complex_types
```

Si executam directament la sentència següent, veurem que no ens retorna cap resultat:

```
SELECT * from persones;
```

El motiu és que la sintaxi de les consultes sobre tipus complexos és una mica diferent que en HiveQL. En Impala hem d'incloure sempre com a prefix el nom del STRUCT:

```
SELECT persona.* from persones;
```

Que retorna (igual que ho faria en Hive):

```

+-----+-----+
| nom  | edat |
+-----+-----+
| Joan | 50   |
| Aina | 30   |
| Pep  | 40   |
+-----+-----+

```

Per recuperar el nom de les persones majors de 35, seria la mateixa sentència que en Hive:

```
SELECT persona.nom FROM persones where persona.edat>35;
```

Que retorna:

-----+
persona.nom
-----+
Joan
Pep
-----+

5.2. Un segon exemple

Anem ara a incloure ja els altres dos tipus de dades complexes, ARRAY i MAP. En aquest primer, a partir de les següents dades sobre la reserva de serveis d'hotel:

Reserva_ID	Client_Info	Serveis_Addicionals	Despeses_Per_Categoria
1	{Nom: "Aina Torres", Mail: "atorres@gmail.com"}	["Esmorzar", "Spa"]	{"Esmorzar": 30, "Spa": 50}
2	{Nom: "Joana Serra", Mail: "jserra@gmail.com"}	["Wi-Fi", "Aparcament"]	{"Wi-Fi": 10, "Aparcament": 15}
3	{Nom: "Javier Ruiz", Mail: "ruizjavier@hotmail.com"}	["Esmorzar"]	{"Esmorzar": 25}
4	{Nom: "Pere Riera", Mail: "pereriera@gmail.com"}	["Spa", "Aparcament", "Esmorzar"]	{"Spa": 50, "Aparcament": 20, "Esmorzar": 40}

Vegem que, a més de Reserva_ID que és de tipus (escalar) INT, cada una de les columnes de la taula és d'un tipus complex diferent:

- Client_Info és un STRUCT amb camps per al nom i correu del client.
- Serveis_Addicionals és un ARRAY dels serveis sol·licitats.
- Despeses_Per_Categoria és un MAP de parelles clau-valor que indiquen el cost associat a cada servei.

Comencem creant la taula en Hive:

```
CREATE TABLE reserves (
  Reserva_ID INT,
  Client_Info STRUCT<Nom: STRING, Mail: STRING>,
  Serveis_Addicionals ARRAY<STRING>,
  Despeses_Per_Categoria MAP<STRING, DOUBLE>
)
STORED AS PARQUET;
```

Vegem la descripció de la taula:

```
desc reserves;
```

col_name	data_type	comment
reserva_id	int	
client_info	struct<Nom:string,Mail:string>	
serveis_adicionals	array<string>	
despeses_per_categoria	map<string,double>	

A continuació, encara des de Hive, farem la inserció de les dades. Ja hem vist que per inserir els STRUCT hem emprat la funció NAMED_STRUCT. De la mateixa manera, també tenim les funcions ARRAY i MAP per als tipus complexes corresponents. Vegem les sentències d'inserció:

```

INSERT INTO reserves
SELECT 1,
NAMED_STRUCT('Nom', 'Aina Torres', 'Mail', 'atorres@gmail.com'),
ARRAY('Esmorzar', 'Spa'),
MAP('Esmorzar', 30.0, 'Spa', 50.0);
INSERT INTO reserves
SELECT 2,
NAMED_STRUCT('Nom', 'Joana Serra', 'Mail', 'jserra@gmail.com'),
ARRAY('Wi-Fi', 'Aparcament'),
MAP('Wi-Fi', 10.0, 'Spa', 15.0);
INSERT INTO reserves
SELECT 3,
NAMED_STRUCT('Nom', 'Javier Ruiz', 'Mail', 'ruizjavier@hotmail.com'),
ARRAY('Esmorzar'),
MAP('Esmorzar', 25.0);
INSERT INTO reserves
SELECT 4,
NAMED_STRUCT('Nom', 'Pere Riera', 'Mail', 'pereriera@gmail.com'),
ARRAY('Spa', 'Aparcament', 'Esmorzar'),
MAP('Spa', 50.0, 'Aparcament', 20.0, 'Esmorzar', 40.0);

```

Comencem veient una consulta que recupera totes les dades de la taula:

```

select * from reserves;

```

reserves.reserva_id	reserves.client_info	reserves.serveis_addicionals	reserves.despeses_per_categoria
1	{"Nom": "Aina Torres", "Mail": "atorres@gmail.com"}	["Esmorzar", "Spa"]	
2	{"Nom": "Joana Serra", "Mail": "jserra@gmail.com"}	["Wi-Fi", "Aparcament"]	
3	{"Nom": "Javier Ruiz", "Mail": "ruizjavier@hotmail.com"}	["Esmorzar"]	
4	{"Nom": "Pere Riera", "Mail": "pereriera@gmail.com"}	["Spa", "Aparcament", "Esmorzar"]	

Ja hem vist abans que amb l'operador . podem accedir als camps d'un STRUCT. Per exemple, recuperem el nom dels clients:

```

select client_info.nom from reserves;

```

nom
Aina Torres
Joana Serra
Javier Ruiz
Pere Riera

En el cas dels arrays, podem accedir a cada posició emprant l'índex (començant per 0) entre claudàtors. També podem emprar la funció SIZE per saber el nombre d'elements de l'array. Per exemple, recuperem el nombre de serveis reservats i el primer d'ells per a cada client:


```
select size(serveis_addicionals) as nombre_serveis_addicionals,
       serveis_addicionals[0] as primer_servei
from reserves;
```

nombre_serveis_addicionals	primer_servei
2	Esmorzar
2	Wi-Fi
1	Esmorzar
3	Spa

També podem emprar la funció ARRAY_CONTAINS per saber si un determinat valor és a un array. Per exemple, recuperem els ID de reserva que contenen el servei addicional Esmorzar:

```
select reserva_id from reserves where array_contains(serveis_addicionals, 'Esmorzar');
```

reserva_id
1
3
4

Per accedir al valor d'una clau determinada d'un MAP, posem entre claudàtors la clau. Per exemple, `despeses_per_categoria['Esmorzar']` en la primera fila ens retornaria 30.0. Vegem com recuperar les despeses d'esmorzar de cada reserva:

```
select reserva_id, despeses_per_categoria['Esmorzar'] as preu_esmorzar from reserves;
```

reserva_id	preu_esmorzar
1	30.0
2	NULL
3	25.0
4	40.0

Consultes amb Impala

Passem ara a veure com treballar amb Impala. Igual que en l'exemple anterior, és probable que sigui necessari actualitzar les metadades:

```
INVALIDATE METADATA complex_types.reserves;
```

Ja hem vist com funcionen els tipus STRUCT, emprant l'operador `.` per accedir als seus camps, igual que en HiveQL. Per exemple, la sentència per recuperar el nom dels clients és igual que en HiveQL:

```
select client_info.nom from reserves;
```

```
+-----+
| client_info.nom |
+-----+
| Aina Torres    |
| Joana Serra    |
| Javier Ruiz    |
| Pere Riera     |
+-----+
```

En canvi, els ARRAYS i MAPS funcionen d'una manera diferent en el SQL d'Impala. En el cas dels ARRAY, l'array és com si fos una taula interna que conté dues columnes, item, amb el valor, i pos, amb la posició (començant per 0). Vegem la descripció de reserves.serveis_addicionals:

```
desc reserves.serveis_addicionals
```

```
+-----+-----+-----+
| name | type  | comment |
+-----+-----+-----+
| item | string |         |
| pos  | bigint |         |
+-----+-----+-----+
```

D'aquesta manera, hem d'afegir reserves.serveis_addicionals a la clàusula FROM, com si fos una taula. Vegem com recuperar tots els serveis addicionals i la seva posició per a la primera reserva:

```
select servei.item, servei.pos
from reserves, reserves.serveis_addicionals as servei
where reserva_id=1;
```

```
+-----+-----+
| item   | pos |
+-----+-----+
| Esmorzar | 0   |
| Spa     | 1   |
+-----+-----+
```

Si volem tornar a recuperar el primer servei de cada reserva:

```
select reserva_id, servei.item as primer_servei
from reserves, reserves.serveis_addicionals as servei
where servei.pos=0;
```

```
+-----+-----+
| reserva_id | primer_servei |
+-----+-----+
| 1          | Esmorzar      |
| 3          | Esmorzar      |
| 2          | Wi-Fi         |
| 4          | Spa           |
+-----+-----+
```

I per saber el nombre de serveis de cada reserva, necessitem fer agrupar per reserva_id i emprar count:

```
select reserva_id, count(servei.item) as nombre_serveis_addicionals
from reserves, reserves.serveis_addicionals as servei
group by reserva_id;
```

reserva_id	nombre_serveis_addicionals
2	2
4	3
1	2
3	1

Per recuperar les reserves que tenen esmorzar ho feim amb el WHERE:

```
select reserva_id
from reserves, reserves.serveis_addicionals as servei
where servei.item = 'Esmorzar';
```

reserva_id
1
3
4

Per acabar, en el cas dels MAP, el que tenim és també una taula interna amb dues columnes key i value. Vegem la descripció del MAP despeses_per_categoria:

```
describe reserves.despeses_per_categoria;
```

name	type	comment
key	string	
value	double	

Finalment, per recuperar el preu de l'esmorzar de cada reserva, com hem fet abans amb HiveQL, amb el SQL d'Impala ho farem així:

```
select reserva_id, despesa.value
from reserves, reserves.despeses_per_categoria as despesa
where despesa.key= 'Esmorzar';
```

reserva_id	value
4	40
3	25
1	30

5.3. Càrrega de fitxers JSON (només per a Hive)

En els apartats anteriors hem carregat les dades emprant sentències SQL INSERT. No obstant això, el més habitual és que tinguem les dades en algun tipus de fitxer o base de dades. En concret, és habitual trobar tipus complexos en documents JSON. L'objectiu, per tant, és definir unes taules amb la mateixa estructura que les que hem vist (persones i reserves), que empen tipus complexos, però carregant les dades a partir d'arxius JSON. Veurem dues aproximacions. La primera, en aquesta secció, que només ens servirà per a Hive. La segona sí que ens permetrà executar consultes amb Impala.

La manera més senzilla per fer-ho en Hive és emprar un format SerDe específic per a JSON. En tenim dos molt utilitzats: `org.apache.hive.hcatalog.data.JsonSerDe` i `org.openx.data.jsonserde.JsonSerDe`. El primer que hem de fer és descarregar-ne un d'ells i configurar Hive.

Per descarregar `org.openx.data.jsonserde.JsonSerDe`, hem de descarregar l'arxiu [hive-hcatalog-core-1.1.0.jar](#).

Una vegada descarregat (suposem que el tenim al path `/home/cloudera/impala`), podem afegir-lo a la nostra sessió de Hive, amb la sentència:

```
ADD JAR /home/cloudera/impala/hive-hcatalog-core-1.1.0.jar;
```

En aquest cas, només estarà disponible en aquesta sessió, ja sigui de Hue o de Beeline. Si volem afegir-ho de manera permanent a Hive, hem de copiar l'arxiu JAR al path `/usr/lib/hive/lib`.

Si, en canvi, volem emprar `org.openx.data.jsonserde.JsonSerDe`, hem de descarregar dos arxius JAR: [json-serde-1.3.8-jar-with-dependencies.jar](#) i [json-udf-1.3.8-jar-with-dependencies.jar](#)

A continuació hem d'afegir-los a la sessió amb un `ADD JAR`, o copiar-los a `/usr/lib/hive/lib` perquè estiguin disponibles de forma permanent.

Anem ara a crear el nostre arxiu JSON ([persones.json](#)) amb les persones que hem emprat abans.

```
{"nom": "Joan", "edat": 50}
{"nom": "Aina", "edat": 30}
{"nom": "Pep", "edat": 40}
```

És important seguir aquest format, on cada fila de l'arxiu és un objecte. És l'anomenat format [JSON Lines](#).

Ara farem un directori persones a HDFS i hi posarem allà el nostre fitxer:

```
hdfs dfs -mkdir /user/cloudera/impala/persones
hdfs dfs -put persones.json /user/cloudera/impala/persones
```

Ja podem crear la nostra taula, especificant que el format de fila és un dels SerDe per a JSON i que carregui les dades des del path HDFS `/user/cloudera/impala/persones`:

```
CREATE EXTERNAL TABLE persones2 (
  nom STRING,
  edat INT
)
ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'
STORED AS TEXTFILE
LOCATION '/user/cloudera/impala/persones';
```

En aquest cas, hem definit la taula amb dues columnes, nom i edat, sense emprar un tipus STRUCT. Vegem la descripció de la taula persones2:

```
desc persones2;
```

col_name	data_type	comment
nom	string	from deserializer
edat	int	from deserializer

I ja podem fer-hi consultes amb HiveQL, de la manera habitual:

```
select * from persones2;
```

```
OK
```

persones2.nom	persones2.edat
Anna	25
Joan	30
Maria	28

Anem ara a treballar amb les nostres dades de reserves de serveis d'hotel, a partir d'aquest JSON ([reserves.json](#)):

```
{
  "Reserva_ID": 1,
  "Client_Info": {
    "Nom": "Aina Torres",
    "Mail": "atorres@gmail.com"
  },
  "Serveis_Addicionals": ["Esmorzar", "Spa"],
  "Despeses_Per_Categoria": {
    "Esmorzar": 30,
    "Spa": 50
  }
},
{
  "Reserva_ID": 2,
  "Client_Info": {
    "Nom": "Joana Serra",
    "Mail": "jserra@gmail.com"
  },
  "Serveis_Addicionals": ["Wi-Fi", "Aparcament"],
  "Despeses_Per_Categoria": {
    "Wi-Fi": 10,
    "Aparcament": 15
  }
},
{
  "Reserva_ID": 3,
  "Client_Info": {
    "Nom": "Javier Ruiz",
    "Mail": "ruizjavier@hotmail.com"
  },
  "Serveis_Addicionals": ["Esmorzar"],
  "Despeses_Per_Categoria": {
    "Esmorzar": 25
  }
},
{
  "Reserva_ID": 4,
  "Client_Info": {
    "Nom": "Pere Riera",
    "Mail": "pereriera@gmail.com"
  },
  "Serveis_Addicionals": ["Spa", "Aparcament", "Esmorzar"],
  "Despeses_Per_Categoria": {
    "Spa": 50,
    "Aparcament": 20,
    "Esmorzar": 40
  }
}
```

Copiam aquest arxiu al path HDFS reserves:

```
hdfs dfs -mkdir /user/cloudera/impala/reserves
hdfs dfs -put reserves.json /user/cloudera/impala/reserves
```

I cream la nostra taula reserves2, carregant les dades des del nostre arxiu JSON:

```
CREATE EXTERNAL TABLE reserves2 (
  Reserva_ID INT,
  Client_Info STRUCT<Nom: STRING, Mail: STRING>,
  Serveis_Addicionals ARRAY<STRING>,
  Despeses_Per_Categoria MAP<STRING, DOUBLE>
)
ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'
STORED AS TEXTFILE
LOCATION '/user/cloudera/impala/reserves';
```

Vegem la descripció de la taula:

```
desc reserves2;
```

col_name	data_type	comment
reserva_id	int	from deserializer
client_info	struct<nom:string,mail:string>	from deserializer
serveis_addicionals	array<string>	from deserializer
despeses_per_categoria	map<string,double>	from deserializer

I fem una consulta a totes les files de la taula:

```
select * from reserves2;
```

reserves2.reserva_id	reserves2.client_info	reserves2.serveis_addicionals	reserves2.despeses_per_categoria
1	{"nom":"Aina Torres","mail":"atorres@gmail.com"}	["Esmorzar","Spa"]	{"esmorzar":30.0,"spa":50.0}
2	{"nom":"Joana Serra","mail":"jserra@gmail.com"}	["Wi-Fi","Aparcament"]	{"aparcament":15.0,"wi-fi":10.0}
3	{"nom":"Javier Ruiz","mail":"ruizjavier@hotmail.com"}		["Esmorzar"]
4	{"nom":"Pere Riera","mail":"pereriera@gmail.com"}	["Spa","Aparcament","Esmorzar"]	{"aparcament":20.0,"esmorzar":40.0,"spa":50.0}

Podríem continuar executant qualsevol de les consultes en HiveQL que hem vist en la secció anterior.

Consultes amb Impala

Desafortunadament, Impala no té suport per a cap d'aquests formats SerDe de JSON. Per tant, no podem executar cap consulta SQL sobre persones2 o reserves2.

És per aquest motiu que en la secció següent veurem una altra manera de tractar els arxius JSON.

5.4. Càrrega de fitxers JSON (per a Hive i Impala)

Per poder treballar amb les dades amb Impala hem de seguir un altre enfocament. El que farem és emprar el SerDe de JSON per a carregar les dades. Però ara guardarem en cada fila tot el string corresponent a un objecte JSON d'una persona o una reserva. És a dir, en cada fila de la taula, tindrem una fila de l'arxiu JSON. A continuació, emprarem unes funcions que proporciona HiveQL per treballar amb documents JSON, per anar inserint les dades amb l'estructura de tipus complexos que volem, en una altra taula. Vegem-ho pas per pas.

Començarem per `persones.json` i crearem la taula `persones3`.

```
CREATE EXTERNAL table persones3 ( persona STRING ) LOCATION '/user/cloudera/impala/persones';
```

Fem un SELECT de totes les files:

```
select * from persones3;
```

persones3.persona
{"nom": "Joan", "edat": 50}
{"nom": "Aina", "edat": 30}
{"nom": "Pep", "edat": 40}

Cada fila és un objecte JSON. Amb la funció `get_json_object` podem recuperar elements dins d'aquest objecte JSON. Per exemple, amb `get_json_object(persona, '$.nom')` recuperem el camp `nom`:

```
SELECT get_json_object(persona, '$.nom') AS nom from persones3;
```

nom
Joan
Aina
Pep

Ara crearem una nova taula `persones4`, en format Parquet, amb la mateixa definició que la taula `persones` de la secció 5.1:

```
CREATE TABLE persones4 (
  persona STRUCT<nom: STRING, edat: INT>
)
STORED AS PARQUET;
```

I ara transformam les dades que tenim a `persones3`, emprant `get_json_object`, dins aquesta taula4:

```
INSERT INTO persones4
SELECT
  NAMED_STRUCT(
    'nom', get_json_object(persona, '$.nom'),
    'edat', CAST(get_json_object(persona, '$.edat') AS INT)
  ) AS persona
FROM persones3;
```

Ja podem executar consultes com les que varem veure a la secció 5.1:

```
select * from persones4;
```

persones4.persona
{"nom": "Joan", "edat": 50}
{"nom": "Aina", "edat": 30}
{"nom": "Pep", "edat": 40}

```
SELECT persona.nom FROM persones4 where persona.edat>35;
```

nom
Joan
Pep

El més important és que amb aquesta taula, que empra el tipus STRUCT i no SerDe sí que podem interactuar des d'Impala. Primer recarregam les metadades:

```
INVALIDATE METADATA complex_types.persones4;
```

I podem executar consultes SQL en Impala:

```
SELECT persona.nom FROM persones4 where persona.edat>35;
```

persona.nom
Joan
Pep

Vegem ara com fer-ho amb les reserves dels serveis d'hotel, on no només tenim el tipus STRUCT sinó també ARRAY i MAP. Tornem a Hive i comencem creant la taula reserves3 on cada fila conté tot l'objecte JSON d'una reserva:

```
CREATE EXTERNAL TABLE reserves3 (reserva STRING)
LOCATION '/user/cloudera/impala/reserves';
```

I vegem el contingut de la taula:


```
select * from reserves3;
```

reserves3.reserva
{ "Reserva_ID": 1, "Client_Info": { "Nom": "Aina Torres", "Mail": "atorres@gmail.com"}, "Serveis_Addicionals": ["Esmorzar", "Spa"], "Despeses_Per_Categoria": { "Esmorzar": 30, "Spa": 50 } }
{ "Reserva_ID": 2, "Client_Info": { "Nom": "Joana Serra", "Mail": "jserra@gmail.com"}, "Serveis_Addicionals": ["Wi-Fi", "Aparcament"], "Despeses_Per_Categoria": { "Wi-Fi": 10, "Aparcament": 15 } }
{ "Reserva_ID": 3, "Client_Info": { "Nom": "Javier Ruiz", "Mail": "ruizjavier@hotmail.com"}, "Serveis_Addicionals": ["Esmorzar"], "Despeses_Per_Categoria": { "Esmorzar": 25 } }
{ "Reserva_ID": 4, "Client_Info": { "Nom": "Pere Riera", "Mail": "pereriera@gmail.com"}, "Serveis_Addicionals": ["Spa", "Aparcament", "Esmorzar"], "Despeses_Per_Categoria": { "Spa": 50, "Aparcament": 20, "Esmorzar": 40 } }

Ja sabem que amb `get_json_object` podem obtenir els valors d'un element JSON. Emprant això, podem construir una sentència que ens retorni un STRUCT (mitjançant la funció `NAMED_STRUCT`) amb les dades de `Client_Info`:

```
select NAMED_STRUCT(
    'Nom', get_json_object(reserva, '$.Client_Info.Nom'),
    'Mail', get_json_object(reserva, '$.Client_Info.Mail')
) AS Client_Info
from reserves3;
```

client_info
{ "nom": "Aina Torres", "mail": "atorres@gmail.com" }
{ "nom": "Joana Serra", "mail": "jserra@gmail.com" }
{ "nom": "Javier Ruiz", "mail": "ruizjavier@hotmail.com" }
{ "nom": "Pere Riera", "mail": "pereriera@gmail.com" }

En el cas de l'ARRAY, emprarem la funció `SPLIT` per separar els valors que conté `Serveis_Addicionals`. Però prèviament, li llevarem les cometes dobles i els arrays:

```
select
SPLIT(
    REGEXP_REPLACE(
        REGEXP_REPLACE(get_json_object(reserva, '$.Serveis_Addicionals'),
        '\\[|\\]', ''), -- Elimina els claudàtors
        '"', '' -- Elimina les cometes dobles
    ),
    ',' -- Divideix en funció de les comes
) AS Serveis_Addicionals
from reserves3;
```

serveis_addicionals
["Esmorzar", "Spa"] ["Wi-Fi", "Aparcament"] ["Esmorzar"] ["Spa", "Aparcament", "Esmorzar"]

I per a les `despeses_per_categoria`, crearem un MAP (mitjançant la funció `MAP`), especificant cada un dels possibles claus i valors. Hem de convertir els preus a `DOUBLE` mitjançant un cast. Aquesta seria la sentència:

```
select MAP(
    'Esmorzar', CAST(get_json_object(reserva, '$.Despeses_Per_Categoria.Esmorzar') AS
DOUBLE),
    'Spa', CAST(get_json_object(reserva, '$.Despeses_Per_Categoria.Spa') AS DOUBLE),
    'Wi-Fi', CAST(get_json_object(reserva, '$.Despeses_Per_Categoria.Wi-fi') AS DOUBLE),
    'Aparcament', CAST(get_json_object(reserva, '$.Despeses_Per_Categoria.Aparcament') AS DOUBLE)
    ) AS Despeses_Per_Categoria
FROM reserves3;
```

```
+-----+-----+
|      despeses_per_categoria      |
+-----+-----+
| {"Esmorzar":30.0,"Spa":50.0,"Wi-Fi":null,"Aparcament":null} |
| {"Esmorzar":null,"Spa":null,"Wi-Fi":null,"Aparcament":15.0} |
| {"Esmorzar":25.0,"Spa":null,"Wi-Fi":null,"Aparcament":null} |
| {"Esmorzar":40.0,"Spa":50.0,"Wi-Fi":null,"Aparcament":20.0} |
+-----+-----+
```

Així doncs, ja només ens queda crear una nova taula reserves4, amb la mateixa definició que reserves de l'apartat 5.2, en format Parquet:

```
CREATE TABLE reserves4 (
    Reserva_ID INT,
    Client_Info STRUCT<Nom: STRING, Mail: STRING>,
    Serveis_Addicionals ARRAY<STRING>,
    Despeses_Per_Categoria MAP<STRING, DOUBLE>
)
STORED AS PARQUET;
```

I fem les insercions en reserves4 a partir dels objectes JSON que tenim a reserves3, emprant les sentències que acabam de veure:

```
INSERT INTO TABLE reserves4
```

```
SELECT
    CAST(get_json_object(reserva, '$.Reserva_ID') AS INT) AS Reserva_ID,
    NAMED_STRUCT(
        'Nom', get_json_object(reserva, '$.Client_Info.Nom'),
        'Mail', get_json_object(reserva, '$.Client_Info.Mail')
    ) AS Client_Info,
    SPLIT(get_json_object(reserva, '$.Serveis_Addicionals'), ',') AS Serveis_Addicionals,
    MAP(
        'Esmorzar', CAST(get_json_object(reserva, '$.Despeses_Per_Categoria.Esmorzar') AS
DOUBLE),
        'Spa', CAST(get_json_object(reserva, '$.Despeses_Per_Categoria.Spa') AS DOUBLE),
        'Wi-Fi', CAST(get_json_object(reserva, '$.Despeses_Per_Categoria.Wi-fi') AS DOUBLE),
        'Aparcament', CAST(get_json_object(reserva, '$.Despeses_Per_Categoria.Aparcament') AS DOUBLE)
    ) AS Despeses_Per_Categoria
FROM reserves3;
```

Ja tenim la taula reserves4, definida emprant tipus complexes, sobre la que podrem executar consultes tant en Hive com en Impala.

Anem directament a Impala i recarreguem les metadades:

```
INVALIDATE METADATA complex_types.reserves4;
```

I provem algunes de les consultes que havíem executat en la secció 5.2.

El nom de tots els clients:

```
select client_info.nom from reserves;
```

client_info.nom
Aina Torres
Joana Serra
Javier Ruiz
Pere Riera

El primer servei de cada reserva:

```
select reserva_id, servei.item as primer_servei
from reserves, reserves.serveis_addicionals as servei
where servei.pos=0;
```

reserva_id	primer_servei
1	Esmorzar
3	Esmorzar
2	Wi-Fi
4	Spa

El preu de l'esmorzar de cada reserva:

```
select reserva_id, despesa.value
from reserves, reserves.despeses_per_categoria as despesa
where despesa.key= 'Esmorzar';
```

reserva_id	value
4	40
3	25
1	30

6. Particionament

Tal com ja vàrem introduir en el lliurament de Hive, el particionament és una tècnica fonamental per optimitzar la gestió i l'execució de consultes en grans conjunts de dades emmagatzemats en sistemes distribuïts com Hadoop. A Impala, el particionament permet dividir les taules en fragments més petits en funció d'un o més valors d'una columna, cosa que redueix el volum de dades que cal escanejar durant una consulta.

Una partició és un subconjunt físic de les dades emmagatzemades en el sistema de fitxers subjacent (com HDFS o S3). Les particions es basen en els valors d'una o més columnes i s'emmagatzemen com subdirectoris dins del directori de la taula. Així, Impala només escaneja les particions necessàries per respondre a la consulta gràcies a una poda (*pruning*) de particions, cosa que redueix considerablement els temps d'execució. És a dir, Impala selecciona només les particions rellevants durant una consulta, basant-se en els valors especificats a la clàusula WHERE.

Quan es crea una taula particionada en Impala, cal definir explícitament quines columnes s'utilitzen per particionar les dades. Vegem un exemple, on tenim un conjunt de dades sobre vendes, i volem particionar-lo per l'any:

```
CREATE TABLE sales_partitioned (  
    product_id INT,  
    sales_amount DOUBLE,  
    region STRING  
)  
PARTITIONED BY (year INT)  
STORED AS PARQUET;
```

Quan s'insereixen dades, se'ls hi ha d'assignar una partició, ja sigui de manera estàtica (manualment) o dinàmica (automàticament).

En el cas d'una inserció amb particionament estàtic, ho faríem així:

```
INSERT INTO sales_partitioned PARTITION (year=2023)  
VALUES (101, 500.0, 'Europe'),  
       (102, 700.0, 'Asia');
```

En el cas d'emprar particionament dinàmic, primer hem de definir aquestes dues propietats:

```
SET hive.exec.dynamic.partition=true;  
SET hive.exec.dynamic.partition.mode=nonstrict;
```

I a continuació, ja podem fer les insercions:

```
INSERT INTO sales_partitioned  
VALUES (101, 500.0, 'Europe', 2023),  
       (102, 700.0, 'Asia', 2023);
```

Les particions creades amb Hive són totalment compatibles amb Impala, ja que ambdós sistemes utilitzen el mateix catàleg de metadades. No obstant això, tot i que Impala suporta el particionament, no té suport natiu per al *bucketing*. Recordem que el *bucketing* de Hive consisteix a organitzar les dades en els anomenats *buckets*, una espècie de partició, emprant una funció de *hash*.

En l'exemple anterior, cada valor diferent de l'any es guardarà a un subdirectori diferent. Si tenim diverses columnes per al particionament, cada una (en ordre) crea un nivell de directoris. Per exemple, volem particionar les dades per any, mes i dia:

```
CREATE TABLE sales_partitioned (  
    product_id INT,  
    sales_amount DOUBLE,  
    region STRING  
)  
PARTITIONED BY (year INT, month INT, day INT)  
STORED AS PARQUET;
```

Quan feim la inserció d'una venda del 17/12/2024, les dades es copiaran en el path *<path_de_la_taula>/2024/12/17*.



Evita massa particions petites: si una taula té moltes particions petites, el rendiment pot degradar-se a causa del cost de gestió de metadades i de fitxers.

Tria bé les columnes per particionar: utilitza columnes amb baixa cardinalitat (pocs valors únics).

Fes servir formats columnars: per millorar el rendiment, utilitza els formats Parquet o ORC per a les taules particionades.

CONSELLS

7. Impala i HBase

En aquest lliurament 4 del mòdul de Sistemes de big data hem comentat que Hive i Impala s'integren molt bé amb HBase. Allà hem vist que podem crear una taula externa en el metastore de Hive que referencii a una taula de HBase, de manera que puguem executar consultes amb HiveQL (des d'un client Hive) o SQL (des d'un client Impala) sobre la taula de HBase.

Podeu trobar més informació a la documentació de Hive

(<https://cwiki.apache.org/confluence/display/hive/hbaseintegration>) i a la d'Impala

(https://impala.apache.org/docs/build/html/topics/impala_hbase.html).

8. Bibliografia

Per a més informació sobre Apache Impala, és molt recomanable consultar la seva documentació: https://impala.apache.org/docs/build/html/topics/impala_intro.html

Podeu trobar també una versió en PDF (l'anomenada Apache Impala Guide) a <https://impala.apache.org/docs/build/impala-4.3.pdf>

Un conegut llibre de referència, tot i que ja una mica antic, és aquest:

Getting Started with Impala, de John Russell. Editat per O'Reilly (2014). ISBN: 9781491905722. <https://www.oreilly.com/library/view/getting-started-with/9781491905760/>