Big Data

Datawarehousing con Hive

Argomenti della lezione

- Introduzione ad Hive
- La console di Hive
- Tipi di dato
- Files
- Database
- Tabelle
- Caricamento dati
- Query
- Join

Introduzione ad Hive

Apache hive è un sistema data warehouse per Apache Hadoop. Hive consente di eseguire attività di riepilogo, query e analisi dei dati.

Hive consente di proiettare la struttura su dati principalmente non strutturati. Dopo aver definito la struttura, è possibile usare HiveQL per eseguire una query sui dati anche senza alcuna conoscenza di Java o MapReduce.

La sintassi di HiveQL è simile a SQL. Questo l'ha reso uno strumento facile da approcciare sia per i programmatori, sia per operatori di altri settori che, pur non essendo informatici di professione, spesso hanno già avuto esperienze con SQL.

Hive agisce come un database relazionale ed in questo possiamo trovare una certa assonanza con la scelta di SQL come linguaggio di interrogazione.

Hive permette di manipolare i file presenti su HDFS (o altri provider di storage) mediante dei database “virtuali” che crea in una porzione di spazio, denominata **metastore**. Il metastore viene gestito tramite un database relazionale che di default è Apache Derby, ma può essere sostituito con altri. In esso verranno salvati i metadati che relazionano i file dei repository e le tabelle del metastore.

La console di Hive

Hive mette a disposizione 2 console per l'accesso **Hive CLI** (deprecata) e **Beeline.** Come anticipato la prima console è deprecata a beneficio di Beeline per questo motivo utilizzeremo esclusivamente la seconda.

Per avviarla è sufficiente avviare, nel terminale della nella nostra VM Cloudera, il comando:

$ beeline
Beeline version ….
beeline>

a questo punto saremmo connessi alla console beeline ma, per accedere al server HiveServer2, sarà necessario eseguire l'apposito comando.

beeline> !connect jdbc:hive2://localhost:10000 hive cloudera

Il comando precedente ci consentirà, all'interno della nostra VM cloudera, di collegarci al server Hive. **!connect** è la keyword utilizzata per richiedere la connessione al server, essa è seguita dal protocollo/driver, l'host (che nel nostro caso è localhost) e la porta (di default 10000). I termini **hive** e **cloudera** sono rispettivamente username e password utilizzati nella nostra installazione.

Se necessario è possibile utilizzare il comando precedente per accedere direttamente ad uno specifico database. Ad esempio per accedere direttamente al database contatti potremmo utilizzare il comando:

beeline> !connect jdbc:hive2://localhost:10000/contatti hive cloudera

Da questo momento siamo connessi al database e possiamo eseguire i comandi **HiveQL.** Per uscire dalla console possiamo digitare i comandi **!q** o **!quit**

Tipi di dato

Hive permette l'utilizzo dei seguenti tipi di dato:

- **numeri**: sono gestite diverse tipologie di dato numerico (a seconda del range e della precisione) sia per interi che in virgola mobile;
- **stringhe**
- **data e ora**
- **booleani**
- **binari**: per la manipolazione di di array di byte
- **strutture dati**: array, mappe e strutture.

NUMERI

Hive mette a disposizione le seguenti tipologie di dato numerico:

- **TINYINT**: 1 byte;
- **SMALLINT**: 2 byte;
- **INT** o **INTEGER**: 4 byte;
- **BIGINT**: 8 byte;
- **FLOAT**: 4 byte;
- **DOUBLE**: 8 byte;
- **DECIMAL**: 8 byte.

STRINGHE

Per le stringhe sono messi a disposizione le seguenti tre tipologie di dato:

- **STRING**: accetta sequenze di caratteri alfanumerici racchiusi tra apici singoli o doppi;

- **VARCHAR**: per il quale viene fissato un numero massimo di caratteri;
- **CHAR**: con dimensione fissa, dichiarata in fase di definizione.

DATA E ORA

Per la gestione delle date viene messo a disposizione il formato **DATE** che prevede il seguente formato di memorizzazione: YYYY-MM-DD. In questo caso non viene gestita l'informazione relativa alle ore/minuti.

Per gestire un dati temporali completi dell'indicazione del tempo è possibile utilizzare il tipo **TIMESTAMP**. Il formato di memorizzazione è **YYYY-MM-DD HH:MM:SS.fffffffff** dove possono essere indicate anche le informazioni relative ad ora, minuti, secondi e, dopo il punto, i nanosecondi (opzionali).

Inoltre è possibile utilizzare il tipo **INTERVAL** per definire intervalli di tempo. Questa tipologia sfrutta la parola chiave **INTERVAL** seguita da un'informazione numerica e dall'unità di misura dell'intervallo, da scegliere tra **SECOND**, **MINUTE**, **DAY**, **MONTH** e **YEAR**.

STRUTTURE DATI

In Hive si possono anche strutturare informazioni secondo le principali forme di raccolta di dati: **array** per gestire le sequenze con memorizzazione della posizione, **mappe** per la struttura chiave/valore e **struct** per l'organizzazione di dati in proprietà distinte dal nome.

- Un **ARRAY** è costituito da elementi omogenei pertanto un campo di tale tipologia richiederà l'indicazione del tipo di dato gestito (es.: array<string>);
- Le mappe sfruttano il tipo **MAP** e richiedono la definizione dei tipi di dato della chiave e del valore (es.: map<string, int>);
- Si definiscono con **STRUCT** e impongono di elencare il nome delle proprietà seguito dal tipo di dato che lo contraddistingue (esempio: struct<indirizzo:string, numero_civico:string, citta:string>).

Files

Hive permette di lavorare comodamente su file collocati nel file system di Hadoop. Quando si definiscono tabelle nei database di Hive, devono essere specificati non solo i tipi di dato dei campi ma anche la tipologia di file su cui la struttura dati poggerà.

La scelta di un formato di file impatta in primo luogo sui seguenti tre aspetti:

- ottimizzazione delle prestazioni;
- occupazione di spazio;
- possibilità di evoluzione della struttura dei dati.

Naturalmente non esiste un formato file perfetto per tutte le esigenze. La tipologia di file da utilizzare dovrebbe essere scelto in base alle informazioni che intendiamo immagazzinare ed analizzare.

Uno degli aspetti che va considerato per primo è se riteniamo di avere bisogno di un **immgazzinamento per righe o per colonne**.

Il primo approccio è ideale quando il grosso delle nostre analisi accederà a molti campi di un certo numero di righe (ad es. formato CSV). Il secondo, a colonne, permette un migliore accesso alle colonne pertanto perfetto per elaborazioni che analizzeranno un gran numero di elementi appartenenti ad un set limitato di colonne.

Altro aspetto da considerare è la **divisibilità dei file**, quella proprietà che in inglese viene identificata con il termine splitability. I Big Data lavorano con grandissime quantità di informazioni è spesso necessario di suddividere i file in blocchi da elaborare separatamente.

Infine si deve pensare alla **possibilità di evoluzione della struttura dei dati**.

Con Hive è possibile utilizzare le seguenti tipologie di files:

- **file di testo**: si tratta del formato di default che prevede l'uso di file di testo semplici, strutturati a piacimento, leggibili per righe;
- **sequence file**: sono file di natura binaria in cui i dati vengono inseriti in coppie chiave/valore disposte in sequenza. Non contengono metadati al loro interno e supportano la compressione a blocchi. Più che per la conservazione di dati sono spesso usati per lo scambio di informazioni tra fasi dello stesso processo (ad esempio, tra gli step del **MapReduce**);
- **ORC file**: si tratta di Optimazed Row Columnar quindi una versione ottimizzata della concezione riga-colonna. Supporta i tipi di dato previsti da Hive, prevede compressione a livello di blocco, permette letture concorrenti e stabilisce limiti al consumo di memoria nelle operazioni di lettura e scrittura. Ha elevate prestazioni nelle operazioni di analisi ed è uno dei formati più adatti ad Hive;

- **RC File**: sono file binari dove gli elementi sono conservati come coppie chiave/valore. Sono nati per ottimizzare lavori svolti su **MapReduce**. La sigla RC sta per row columnar in quanto questo formato riesce a trarre sia i vantaggi dell'immagazzinamento per righe sia di quello per colonne. Questo approccio permetterà contemporaneamente di velocizzare il caricamento di dati e l'esecuzione di interrogazioni, evitando recuperi di colonne non necessarie. I limiti del RC vertono sulle prestazioni in scrittura non alte e sulla invariabilità della struttura dei dati nella maggior parte dei tool che lo usano: tipicamente per variare lo schema di un file RC è necessario riprocessarlo;
- **Avro**: è un altro formato nato appositamente per Hadooop che minimizza lo spazio di archiviazione dei dati. All'interno di un file di questo tipo esiste una porzione di intestazioni ed una di dati. Quella di intestazione è strutturata con JSON e specifica, tra l'altro, protocolli e tipi di dato impiegati nel file: ciò rende i file Avro i più adatti in assoluto all'evoluzione della struttura interna del file. Inoltre si tratta di un formato "a righe", che supporta la compressione ed è particolarmente adatto ad operazioni write-intensive facilitando le operazioni di scrittura;

- **Parquet**: file di questo tipo sono molto utilizzati al giorno d’oggi in framework per Big Data. Hanno un approccio del tutto colonnare pertanto sono caratterizzati da una buona compressione dei dati e prestazioni notevoli in fase di analisi. I file contengono metadati e le informazioni sono organizzate in blocchi ognuno dei quali ha una struttura colonnare.

Database

Hive utilizza dei database, ma non è a tutti gli effetti un DBMS relazionale. Sfrutta il modello relazionale come paradigma per l'accesso ai file collocati sui sistemi di storage con cui lavora.

Possiamo vedere quanti database contiene l'installazione di Hive eseguendo il comando all'interno della console di Hive:

SHOW DATABASES;

Possiamo creare un nuovo database con il comando:

CREATE DATABASE nome_database;

questo comando può essere integrato con delle informazioni aggiuntive, come un commento ed una serie di properties nel formato chiave/valore.

```
CREATE DATABASE IF NOT EXISTS contatti
  COMMENT "Database per la gestione dei contatti"
  WITH DBPROPERTIES ("autore"="DND","creato"="1/11/2020");
```

Per ottenere delle informazioni sul un database presente possiamo utilizzare il comando:

```
DESCRIBE DATABASE nome_database;
```

Se necessario è possibile effettuare delle modifiche ad un database presente in Hive con i seguenti comandi:

- **ALTER DATABASE nome_database SET DBPROPERTIES (“prop”=”val”)**: modifica delle proprietà del database;

- **ALTER DATABASE nome_database SET OWNER**: modifica dell'utente proprietario;
- **ALTER DATABASE nome_database SET LOCATION**: modifica della collocazione su HDFS.

Per cancellare una database, si usa **DROP DATABASE**. Nel caso in cui si richieda di eliminare un database non esistente viene restituito un errore che si può evitare con il modificatore **IF EXISTS**

DROP DATABASE IF EXISTS nome_datababse;

Un altro modificatore essenziale è **CASCADE**. Hive, di base, può cancellare solo database vuoti, ma aggiungendo CASCADE in coda al comando verrà eliminato insieme a tutto il suo contenuto.

DROP DATABASE nome_database CASCADE;

Una volta creato un database, potremo creare e modificare oggetti al suo interno. Se ad esempio vorremo creare la tabella dipendenti nel database archivio dovremo indicare archivio.dipendenti sia al momento della creazione sia di ogni altra invocazione. Alternativamente, è possibile indicare il database corrente in una sessione di lavoro utilizzando la parola chiave **USE**

USE nome_database;

ogni tabella nominata nei comandi successivi verrebbe riferita a tale database.

Tabelle

All’interno di un database definiamo tabelle rappresentate da un file collocato nello spazio di storage gestito da Hive. La tabella potrà agganciarsi ad un file già esistente o ad uno che essa stessa creerà con le operazioni di inserimento. Il comando che crea una tabella è **CREATE TABLE** e richiede obbligatoriamente il nome della tabella da creare.

```
CREATE TABLE impiegati (
 matricola STRING,
 cognome STRING,
 stipendio INT,
 data_assunzione DATE,
 dipartimento STRING
) STORED AS PARQUET;
```

nel comando precedente possiamo notare la somiglianza al costrutto di creazione di una tabella in SQL.

Abbiamo definito il nome, i campi con i relativi tipi ed, infine, il tipo di file che deve essere utilizzato per il salvataggio dei dati.

L'indicazione di **STORED AS** non è obbligatoria. Quando non è fornita, indica che il tipo di file sarà quello di default, ovvero **TEXTFILE**

Una volta creata la tabella è possibile controllare se è stata creata con il comando:

SHOW TABLES;

che mostra la lista di tabelle presenti nel database corrente. E' inoltre possibile richiedere delle informazioni di dettaglio sulla tabella con il comando:

DESCRIBE nome_tabella;

Con il seguente comando potremmo anche creare anche una tabella per gestire un file CSV.

```
CREATE TABLE impiegati (
 matricola STRING,
 cognome STRING,
 stipendio INT,
 data_assunzione DATE,
 dipartimento STRING
)ROW FORMAT DELIMITED
FIELDS TERMINATED BY ';';
```

a questo punto provando ad effettuare delle insert per popolare la tabella verrebbe creato, su HDFS, per la gestione del contenuto della tabella.

```
INSERT INTO impiegati VALUES ('101','Sili', 60, '1978-12-03', ‘NO’), ('102', ‘Rossi’,40, '1990-07-21', ‘NO’), ('103', 'Neri', 40, '1980-08-20', ‘NO’);
```

La creazione di una tabella può essere effettuata come copia di una tabella esistente. Esistono due modi per effettuare la copia di una tabella:

Creando solo la struttura della nuova tabella copiandola dalla tabella sorgente (non vengono copiati i dati):

CREATE TABLE tabella_destinazione LIKE tabella_sorgente;

Oppure possiamo copiare anche i dati con la sintassi:

CREATE TABLE tabella_destinazione AS SELECT * FROM tabella_sorgente;

Altra possibilità che ci viene offerta nella creazione di una tabella è la possibilità di creare delle tabelle che memorizzano i dati in una posizione differente da quella che Hive utilizza di default per lo storage. Questa evenienza è utile quando vogliamo

incorporare in Hive dati esterni senza effettuare copie/spostamenti oppure quando i dati sono condivisi con altre applicazioni.

Per farlo dobbiamo usare la parola chiave **EXTERNAL** e indicare con **LOCATION** quale collocazione di HDFS ne costituirà il contenuto: attenzione che l'indirizzo deve puntare ad una cartella, mai ad un file.

```
CREATE EXTERNAL TABLE transazioni(
 data DATE,
 importo DECIMAL,
 codice STRING,
 descrizione STRING,
 id_conto INT
)ROW FORMAT DELIMITED
FIELDS TERMINATED BY ';'
LOCATION 'hdfs://localhost:9000/banca';
```

Per eliminare il contenuto di una tabella possiamo utilizzare il comando:

TRUNCATE TABLE nome_tabella;

Se invece volessimo cancellare proprio la tabella dal database dovremmo utilizzare il comando:

DROP TABLE nome_tabella;

Per eliminare il contenuto di una tabella possiamo utilizzare il comando:

TRUNCATE TABLE nome_tabella;

Se invece volessimo cancellare proprio la tabella dal database dovremmo utilizzare il comando:

DROP TABLE nome_tabella;

Caricamento dati

Hive dispone di meccanismi per il caricamento dei dati sia singoli che massivi. Inoltre permette di effettuare operazioni di export delle informazioni.

L'inserimento di uno o più record all'interno delle nostre tabelle può essere effettuato tramite l'esecuzione di una INSERT come la seguente:

INSERT INTO impiegati VALUES ('101','Di Nuzzo', 60, '1985-03-07', 'NO');

L'inserimento può essere multiplo specificando più tuple di dati, separate da virgole, dopo la parola chiave VALUES.

Considerando lo scopo ed il contesto in cui si utilizza Hive l'inserimento di record puntuali non è così frequente.

Un tipo di operazione molto più comune è l'importazione massiva di dati in una tabella da file. Per effettuare un import massivo viene utilizzato il comando **LOAD DATA**.

Considerando l'ultima versione della tabella impiegati create potremmo caricare massivamente i dati da un file CSV locale con il seguente comando:

LOAD DATA LOCAL INPATH 'dati_impiegati.csv' INTO TABLE impiegati;

Il file è stato individuato su percorso locale ma eliminando la parola chiave **LOCAL** se ne cercherà la presenza in HDFS.

Infine è possibile utilizzare i comandi **EXPORT TABLE** ed **IMPORT TABLE** per gestire operazioni di passaggio di dati tra tabelle e database differenti.

Il comando **EXPORT TABLE** ci consente di esportare i dati inseriti in una tabella in una cartella collocata nel sistema HDFS.
Supponiamo, ad esempio, di voler estrarre in un file CSV tutti i dati inseriti in una determinata tabella

EXPORT TABLE nome_database.nome_tabella TO '/tmp/copia';

Si noti che l'indirizzo si riferisce ad un percorso interno a HDFS che deve essere racchiuso tra apici. Inoltre il percorso deve indicare una cartella, non un file. Questa sarà collocata in HDFS e conterrà a sua volta una sottocartella data in cui sarà salvato il file csv.

Analogamente possiamo importare in una tabella tutti i file contenuti in una cartella mediante la direttiva **IMPORT TABLE**

IMPORT TABLE nome_tabella FROM '/tmp/copia';

Query

Se volete evitare di scrivere ogni volta il codice SQL per i test potete creare un script SQL ed eseguirlo interamente con il comando:

hive -f nome_file.sql

dal terminale della vostra macchina virtuale.

Per recuperare tutte le informazioni contenute in una determinata tabella possiamo utilizzare la query:

SELECT * FROM impiegati;

Come in SQL potrebbe ridurre il numero di colonne o personalizzarne la posizione con la sintassi:

SELECT matricola, cognome FROM impiegati;

Se necessario è possibile utilizzare la clausola DISTINCT per richiedere una lista di valori non duplicati:

SELECT DISTINCT stipendio FROM impiegati;

WHERE

Con la clausola WHERE possiamo specificare condizioni di selezione delle singole righe. Lo scopo di ciò è attuare filtri in maniera da poter scegliere quali righe della tabella possono far parte del set di risultati.

SELECT * FROM impiegati WHERE stipendio>40

Gli **operatori di confronto** che possono essere utilizzati sono quelli consueti del linguaggio SQL: oltre al maggiore (>), abbiamo il minore(<), minore o uguale (<=), maggiore o uguale (>=) e uguale (=).

Per legare più confronti tra loro è possibile utilizzare i seguenti operatori logici: **AND** che restituisce TRUE solo se entrambi i suoi operandi sono veri; **OR** che per restituire TRUE richiede che almeno uno dei suoi operandi sia vero; **NOT** (indicabile anche con il punto esclamativo) inverte il valore booleano che lo segue.

A questi operatori si può aggiungere **IN** utilizzabile nel formato A IN (valore1, valore2, valore3, ...) il quale restituisce TRUE se A equivale a uno dei valori rappresentati tra parentesi.

All'operatore **IN** si può applicare la negazione del **NOT** pertanto A NOT IN (valore1, valore2, valore3, ...) sarà TRUE se A non equivale a nessuno dei valori indicati tra parentesi.

LIMIT

Allo scopo di ridurre il set di risultati di un'interrogazione si può usare la clausola LIMIT che determina quanti record al massimo una query può restituire.

SELECT * FROM impiegati WHERE stipendio>40 LIMIT 1

In questo modo verrà restituito un solo risultato.

E' anche possibile utilizzare la clausola limit per effettuare una sorta di paginazione dei risultati:

SELECT * FROM impiegati WHERE stipendio>40 LIMIT 0, 10

In questo modo richiediamo di vedere 10 risultati a partire dal primo. Utilizzando LIMIT 10, 10 avremmo richiesto di vedere 10 risultati a partire dal decimo.

Durante le operazioni di query, si possono richiedere velocemente calcoli ed elaborazioni sfruttando gli operatori e le incluse nella piattaforma Hive.

operatori aritmetici: includono tutto ciò che serve per svolgere le tipiche operazioni aritmetiche. In primis, ci sono i simboli +, -, * e / che, rispettivamente, rappresentano le operazioni di somma, sottrazione, moltiplicazione e divisione. A questi si aggiungono l'operatore DIV che restituisce la parte intera del risultato di una divisione (17 DIV 3 restituisce 5), e % che fornisce il resto di una divisione (17 % 3 risulta 2);

operatori bitwise: una categoria di operatori aritmetici che permettono di svolgere le operazioni bit a bit. Troviamo & per l'operazione di AND (solo se due bit corrispondenti sono entrambi pari a 1, il bit risultato sarà 1 altrimenti sempre zero: 12 & 9 = 8). L'operatore OR è rappresentato con | e determina che un solo bit a 1 tra i due è sufficiente per ottenere 1 come risultato (12 | 9 = 13), mentre lo XOR si rappresenta con ^ e fornisce 1 come risultato solo se gli operandi sono una coppia di bit di valore diverso (12 ^ 9 = 5). Con il simbolo ~ si ottiene l'operatore NOT che inverte il valore dei bit in un dato (~8 = -9);

operatore di concatenazione per le stringhe: il simbolo || può essere utilizzato a tale scopo, in alternativa alla funzione CONCAT che vedremo nel seguito della lezione.

funzioni di aggregazione: COUNT, MIN, MAX, AVG e SUM per le operazioni matematiche. variance, var_pop e var_samp per la varianza, stdev_pop e std_samp dedicate alla deviazione standard, percentile per i percentili, corr per la correlazione nonché regr_intercept, regr_slope e regr_r2 per quanto riguarda la regressione lineare.

manipolazione data/ora: YEAR, MONTH e DAY che recuperano da un campo di tipo DATE i valori dell'anno, del mese e del giorno. Esistono anche hour, minute e second per ore, minuti e secondi, datediff e date_add per l'aritmetica tra date oppure current_date e current_timestamp per ottenere data attuale e timestamp attuale.

funzioni per le stringhe: CONCAT per concatenare, substr per ottenere una sottostringa, lower per ottenere la versione minuscola di una stringa (esiste anche upper per il maiuscolo)

funzioni condizionali: IF(ESPRESSIONE, SE_TRUE, SE_FALSE) possiamo valutare una condizione e definire quale valore sarà restituito in caso questa sia TRUE e quale in caso contrario. Per valori specifici ma più variegati si può chiamare in causa il costrutto CASE...WHEN...THEN...END, ad esempio CASE campo WHEN 'valore1' THEN 'valore_restituito' WHEN 'valore2' THEN 'restituito_in_caso_2' END

ORDINAMENTO

E’ possibile ordinare il risultato di una query in base ad uno o più campi con il costrutto ORDER BY (in ordine crescente, default, o decrescente con DESC)

RAGGRUPPAMENTO

Per eseguire il raggruppamento si sfrutta la clausola **GROUP BY**, anch'essa seguita dai campi in base ai quali vogliamo procedere al raggruppamento. Raggruppare significa essenzialmente suddividere i record in gruppi diversi, ognuno dei quali contraddistinto da medesimi valori nei campi indicati dal GROUP BY.

Join

Hive, nonostante sia applicato sui Big Data, ha una mentalità pienamente relazionale pertanto permette di fare uso di Join e mette a disposizione l'apposita parola chiave JOIN.

Considerando la presenza delle tabelle presenti nel seguente schema

impiegato

Matricola	Cognome	Stipendio	Dipartimento
101	Sili	60	NO
102	Rossi	40	NO
103	Neri	40	NO
201	Neri	40	SU
202	Verdi	50	SU
301	Bisi	70	IS

dipartimento

Codice	Nome	Sede	Direttore
NO	Nord	Milano	101
SU	Sud	Napoli	201
IS	Isole	Palermo	301

progetto

Sigla	Nome	Bilancio	Responsabile
Alpha	Vendite	30	202
Beta	Inventario	50	301
Gamma	Distribuzione	18	301

partecipazione

Impiegato	Progetto
101	Alpha
101	Beta
103	Alpha
103	Beta
201	Beta
202	Beta

Hive ci permette di effettuare operazioni di JOIN per relazionare le informazioni tra impiegato e dipartimento nel seguente modo:

SELECT i.matricola, i.cognome, d.sede
FROM impiegato as i JOIN dipartimento as d ON i.Dipartimento = d.Codice

è possibile anche effettuare operazioni di LEFT JOIN.

Hive ci mette a disposizione anche la possibilità di creare delle VIEW come in un normale database relazionale.

CREATE VIEW IF NOT EXISTS conti_view AS
SELECT ….

possiamo modificare la VIEW con **ALTER VIEW nome_view as** o eliminarla con **DROP VIEW nome_view**

Come in SQL anche Hive consente l’utilizzo di subquery.