Partie 2
Hive: Distributed Data Warehouse

1. Aperçu et architecture
2. Data Definition Language (DDL)
3. Data Manipulation Language (DML)
4. Data Query Language (DQL)

Ressources :

https://hive.apache.org/

https://cdoss.tech/

https://e.huawei.com/en/talent/#/

Aperçu et architecture

- Hive est un outil d'entrepôt de données permettant de traiter des données et des requêtes distribuées dans Hadoop.
- Il permet de lire, écrire et gérer un grand volume de données résidant dans un stockage distribué.
- Il permet de faciliter la définition des requêtes et l'analyse de données.
- Il a été développé par Facebook, puis a été repris par l'Apache Software Foundation qui l'a développé en open source sous le nom d'Apache Hive.

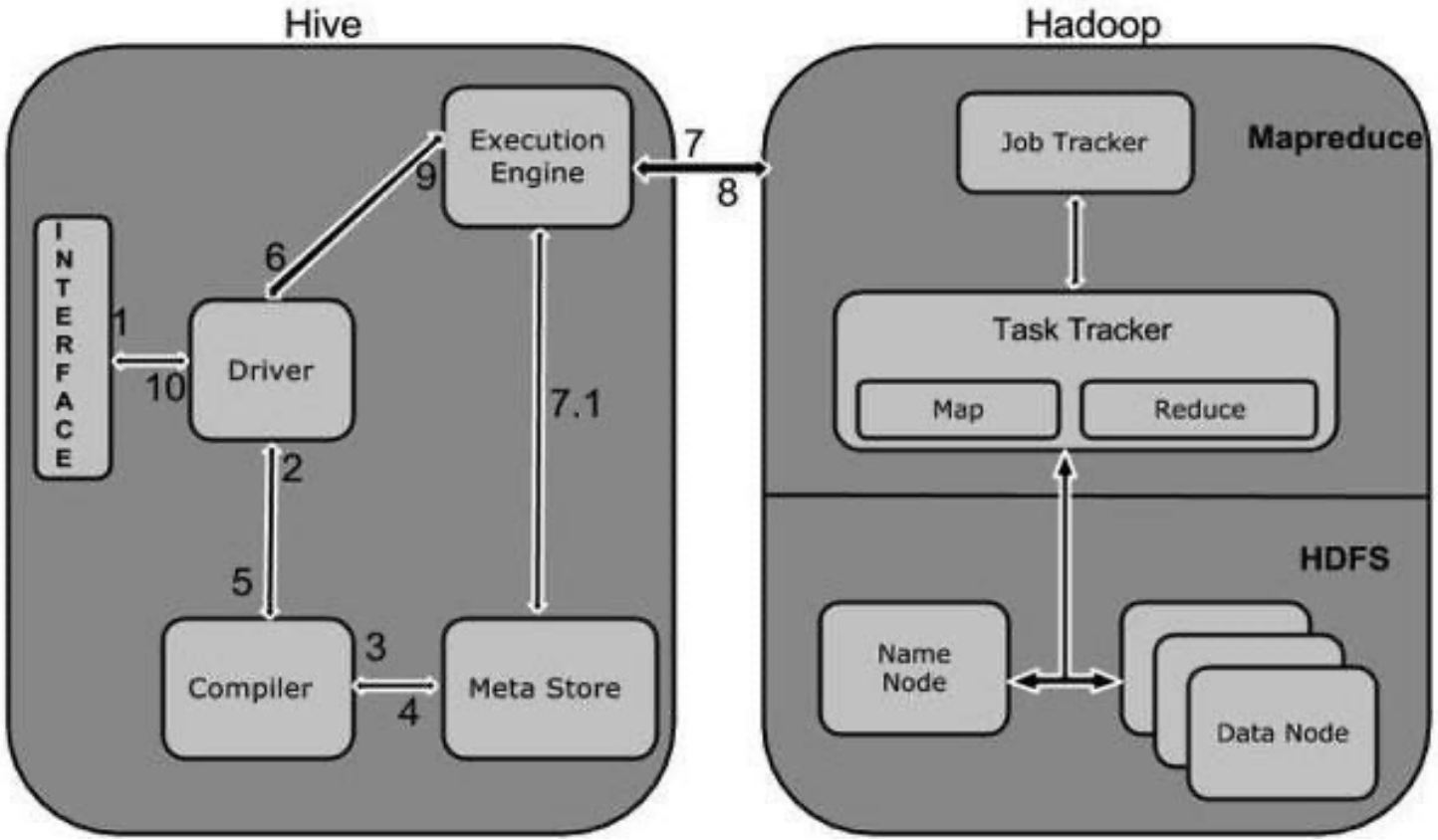
Aperçu et architecture

- Caractéristiques de Hive :
  - Permet un accès facile aux données via SQL
  - Facilite des tâches du Data Warehouse telles que l'extraction/la transformation/le chargement (ETL) et l'analyse de données.
  - Supporte l'accès directe aux fichiers HDFS et Hbase.
  - Supporte l'exécution de requêtes vis les moteurs Apache Tez ou Spark.
  - Facile à utiliser.

  - *Hive **n'est pas** une BD relationnelle.*
  - *Hive **n'est pas** un langage de requêtes Temps-Réels.*

Aperçu et architecture

Processus d'exécution de requête :



Aperçu et architecture

1. l'interface Hive, Command Line ou bien Web UI, envoie une requête au Driver.
2. Le Driver envoie la requête au compilateur.
3. Le compilateur vérifie la syntaxe et demande les méta-données du Metastore
4. Metastore envoie les méta-données au compilateur.
5. Le compilateur envoie le plan d'exécution au Driver.
6. Le Driver envoie le plan d'exécution au Execution Engine.

Aperçu et architecture

7. Puisque le processus d'exécution est un MapReduce, l'Exécution Engine envoie le job au JobTracker du Name Node. Ce dernier attribue le job au TaskTracker qui se trouve dans le Data Node.
8. Le Exécution Engine reçoit le résultat.
9. Le Exécution Engine envoie le résultat au Driver.
10. Le Driver envoie le résultat à l'interface Hive.

Partie 2
Hive: Distrubuted Data Warehouse

1. Aperçu et architecture
2. Data Definition Language (DDL)
3. Data Manipulation Language (DML)
4. Data Query Language (DQL)

Data Definition Language (DDL)

- Data Types :

Numeric Types	Description
TINY INT	Entier sur 1 byte (-128 to 127)
SMALL INT	Entier sur 2 bytes (-32768 to 32767)
INT	Entier sur 4 bytes
BIG INT	Entier sur 8 bytes
FLOAT	Réel sur 4 bytes
DOUBLE	Réel sur 8 bytes
DECIMAL	Le nombre de chiffre à droite et à gauche de la virgule est à préciser

String Types	Description
CHAR	Chaine sur 265 max
VARCHAR	Chaine sur 65355 max
STRING	Chaine de longueur 2G max

Date Type	Description
TIMESTAMP	Supporte le timestamp traditionnel de l'Unix
DATE	Date avec le format YYYY-MM-DD.

Data Definition Language (DDL)

- Hive peut définir des bases de données pour stocker des données structurées sous forme de tables, puis passer des requêtes pour les analyser.
- Hive contient une base de données par défaut, nommée ***default***.
- Création de base de données :

```
CREATE DATABASE|SCHEMA [IF NOT EXISTS] <database name>
```

- ***Create Database*** permet de créer une BD Hive. Une BD est une collection de tables.
- Nous pouvons utiliser ***SCHEMA*** ou bien ***DATABASE***.

Data Definition Language (DDL)

- La clause optionnelle ***IF NOT EXISTS*** permet d'éviter le message d'erreur si une BD avec le même nom existe déjà.
- Exemple :

```
CREATE DATABASE studentsdb;
```

- L'instruction `SHOW DATABASES` permet d'afficher la liste des BD.
- Pour utiliser une BD : `USE database_name;`
- Pour vérifier la BD actuelle : `SELECT current_database()`

Data Definition Language (DDL)

- Suppression d'une base de données :

```
DROP (DATABASE|SCHEMA) [IF EXISTS] database_name [RESTRICT|CASCADE];
```

- La clause optionnelle ***IF EXISTS*** permet d'éviter le message d'erreur si la BD n'existe pas.
- Le comportement par défaut est ***RESTRICT***, avec lequel la suppression échoue si la BD n'est pas vide.
- L'option ***CASCADE*** permet de supprimer les tables avant de supprimer la BD correspondante.
- Exemple :

```
DROP DATABASE IF EXISTS studentsdb CASCADE;
```

Data Definition Language (DDL)

- Création d'une table :

```
CREATE [TEMPORARY] [EXTERNAL] TABLE [IF NOT EXISTS][db_name.]table_name
[(col_name data_type [COMMENT col_comment], ...)]
[COMMENT table_comment]
[PARTITIONED BY (col_name data_type, ...)]
[CLUSTERED BY (col_name, col_name, ...)
      [SORTED BY (col_name [ASC|DESC], ...)] INTO num_buckets BUCKETS]
[ROW FORMAT row_format]
[STORED AS file_format]
[LOCATION hdfs_path]
[AS select_statement]
```

Data Definition Language (DDL)

- La clause ***IF NOT EXISTS*** permet d'éviter le message d'erreur si une table avec le même nom existe déjà.
- La clause `Temporary` indique que la table persiste seulement Durant la session. Elle sera supprimée à la fin de la session.
- `db_name` permet d'indiquer la BD concernée.
- Une table interne est enregistrée par défaut dans le répertoire warehouse de Hive :

  `/user/hive/warehouse/databasename.db/tablename/`
- Cependant, l'emplacement d'enregistrement par défaut peut être modifié par la clause LOCATION.
- Un répertoire sera créé pour chaque table.

Data Definition Language (DDL)

- Hive traite deux types de structures de table : interne (par défaut) et externe, indiquée par la clause ***EXTERNAL***.
- Si une table interne est supprimée, ses données et métadonnées seront également supprimées.
- Une table interne peut être créée :
  - pour un fichier existant disponible sur HDFS. Les données seront chargées par la suite du fichier vers la table, ou bien.
  - Sans correspondance avec un fichier existant. Les données seront insérées avec INSERT
- Utiliser une table interne lorsque tout le cycle de vie de la table est géré par Hive.

Data Definition Language (DDL)

- Une table externe est créée pour un fichier existant disponible sur HDFS.
- Une table externe peut être accessible par d'autres systèmes, à l'extérieur de Hive.
- Si une table externe est supprimée, seules ses métadonnées seront supprimées.
- Utiliser une table externe lorsque
  - le fichier de données existe déjà et lorsque ce fichier doit exister même si la table est supprimée.
  - Ce fichier peut être utiliser par d'autres systèmes

Data Definition Language (DDL)

- `[COMMENT col_comment], ...)] / [COMMENT table_comment]` permettent de commenter une colonne / table.
- `[ROW FORMAT DELIMITED FIELDS TERMINATED BY ',']` : indique que les colonnes dans le fichier source (à partir duquel les données seront chargées) sont séparées par ',’. Par défaut, le séparateur est le caractère control+A.

- `CREATE TABLE students (name STRING, gender STRING, moy DECIMAL(2, 2)) ROW FORMAT DELIMITED FIELDS TERMINATED BY ‘,’`

- ```
  CREATE EXTERNAL TABLE students
        (name STRING, gender STRING, moy DECIMAL(2, 2))
        ROW FORMAT DELIMITED FIELDS TERMINATED BY ‘,’
        LOCATION ‘/user'
  ```
  ```

Data Definition Language (DDL)

- `[ROW FORMAT SerDe …]` : Hive utilise l'interface SerDe pour les E/S. SerDe permet à Hive de lire et écrire les données d'une table dans un format personnalisé :
- `ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.RegexSerDe' WITH SERDEPROPERTIES ( "input.regex" = "<regex>" )` : permet de convertir les données selon une expression régulière.
- `ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'` : permet de convertir les données en format CSV.
- `CREATE TABLE students (name STRING, gender STRING, moy DECIMAL(2, 2)) ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'`

Data Definition Language (DDL)

- `[STORED AS TEXTFILE|PARQUET|AVRO|…]` : permet d'indiquer le format d'enregistrement. Un bon choix du format d'enregistrement permet de faciliter/accélérer la lecture/écriture par différents outils.
- Le format `TEXTFILE` est le format par défaut. Il est facilement compréhensible par plusieurs outils et est caractérisé par le fait que la lecture humaine est possible, mais les performances ne sont pas bonnes.
- Le format `PARQUET` est facilement compréhensible par plusieurs outils et est caractérisé par une bonne performance, mais la lecture humaine est non possible.

Data Definition Language (DDL)

- Le format `PARQUET` est un format de stockage orienté colonne et les valeurs de chaque colonne sont stockées ensemble.
- Le format `AVRO` est facilement compréhensible par plusieurs outils et est caractérisé par une bonne performance, mais la lecture humaine est non possible.
- Le format `AVRO` est un format de stockage orienté ligne.
- `AVRO` stocke le schéma en format JSON permettent une lecture facile par n'importe quel langage.

Data Definition Language (DDL)

- PARQUET est approprié dans le cas où des requêtes demandent certaines colonnes d'une tables formées par plusieurs colonnes.
- AVRO est idéale dans le cas des requêtes où on demande toutes les colonnes.
- AVRO est plus approprié dans le cas où le schéma évolue : ajout ou modification de colonne.

- `CREATE TABLE students (name STRING, gender STRING, moy DECIMAL(2, 2))STORED AS AVRO`

Data Definition Language (DDL)

- Le format `ORC` (Optimized Row Columnar) est un format de stockage orienté colonne.
- Le format `ORC` est compréhensible par moins d'outils que PARQUET et AVRO et est caractérisé par une bonne performance, mais la lecture humaine est non possible.
- Par rapport au format PARQUET, ORC
  - est compréhensible par moins d'outils.
  - Moins approprié pour les données imbriquées.
  - Possède un taux de compression Meilleur.

Data Definition Language (DDL)

- `[PARTITIONED BY (col_name data_type, ...)]` : cette clause permet de diviser les données d'une table en se basant sur une ou plusieurs colonnes.
- La colonne de partition figure seulement dans la clause `PARTITIONED BY`.
- La valeur de la colonne de partition ne sera pas stockée dans le fichier de données.
- Un répertoire sera créé pour chaque valeur de la colonne de partition.
- Ceci permet d'améliorer les performances des requêtes.

Data Definition Language (DDL)

- `[CLUSTERED BY (col_name, col_name, ...)`

  `[SORTED BY (col_name [ASC|DESC], ...)] INTO num_buckets BUCKETS]` : En plus de la partition, les données d'une table ou d'une partition peuvent être séparées (bucketed) en utilisant la clause CLUSTERED BY, et elles peuvent être triées dans le bucket via la clause SORT BY.
- Il faut indiquer le nombre de buckets.
- Un fichier sera créé pour chaque bucket.
- Ceci permet d'améliorer encore plus les performances des requêtes.

Data Definition Language (DDL)

- CREATE TABLE students (name STRING, city STRING, moy DECIMAL(2, 2))
  PARTITIONED BY (gender STRING)
  ROW FORMAT DELIMITED FIELDS TERMINATED BY ‘,’
  STORED AS TEXTFILE

- CREATE TABLE students (name STRING, city STRING, moy DECIMAL(2, 2))
  PARTITIONED BY (gender STRING)
  CLUSTERED BY(city) SORTED BY(moy) INTO 24 BUCKETS
  ROW FORMAT DELIMITED FIELDS TERMINATED BY ‘,’
  STORED AS TEXTFILE

Data Definition Language (DDL)

- `[AS select_statement]` : une table peut être créée et remplie par le résultat d'une requête en une seule instruction (CTAS).
- La partie create de l'instruction CTAS aura le même schéma de la partie select.
- La table créée peut avoir d'autres propriétés définies dans les clauses SerDe et STORED AS.
- La table créée ne peut être que interne.

- ```
  CREATE TABLE studentsMale AS SELECT * FROM students
                                WHERE gender='Male'
  ```
  ```

Data Definition Language (DDL)

- ```
  CREATE [TEMPORARY] [EXTERNAL] TABLE [IF NOT EXISTS]
  [db_name.]table_name
   LIKE existing_table_name
   [LOCATION hdfs_path];
  ```
- La clause LIKE permet de cloner la définition d'une table existante (copie exacte), sans copier les données.
- La nouvelle table est vide.

- ```
  CREATE TABLE studentsFemale LIKE studentsMale
  ```

Data Definition Language (DDL)

- `SHOW TABLES;` : affiche la liste des tables.
- `DESCRIBE table_name;` : affiche une description de la table.
- `DESCRIBE FORMATTED table_name;` : affiche une description plus détaillée de la table.

Data Definition Language (DDL)

- Remarque : Si la première ligne du fichier à partir duquel les données seront chargées représente des titres de colonnes, alors cette ligne doit être ignorée lors du chargement des données :
- `CREATE TABLE table_name (…) TBLPROPERTIES ('skip.header.line.count'='1');`
- Remarque : Hive considère par défaut le caractère \N pour représenter les valeurs nulls.
- Si dans le fichier, le null est représenté autrement (chaine vide: ‘’, null, espace: ‘ ‘, …), alors hive doit être informé :
- `CREATE TABLE table_name (…) TBLPROPERTIES ('serialization.null.format'='');`
- Si la table est déjà créée :
- `ALTER TABLE table_name`

```
SET TBLPROPERTIES ('serialization.null.format' = '');
```

Data Definition Language (DDL)

- Modification de table : l'instruction alter permet de changer le nom d'une table ou d'une colonne, ajouter / supprimer / remplacer une colonne, modifier le type / rang d'une colonne, modifier la propriété d'une table…

```
ALTER TABLE table_name RENAME TO new_name
ALTER TABLE table_name ADD COLUMNS (col_spec[, col_spec ...])
ALTER TABLE table_name DROP [COLUMN] column_name
ALTER TABLE table_name CHANGE column_name new_name new_type
ALTER TABLE table_name REPLACE COLUMNS (col_spec[, col_spec ...])
ALTER TABLE table_name SET TBLPROPERTIES table_properties;
ALTER TABLE table_name ADD PARTITION (partcol1=val1, partcol2=val2 ...);
```

Data Definition Language (DDL)

- Suppression de table : l'instruction DROP permet de supprimer les données et méta-données d'une table interne, et seulement les méta-données d'une table externe.
- `DROP TABLE [IF EXISTS] table_name`

Partie 2
Hive: Distrubuted Data Warehouse

1. Aperçu et architecture
2. Data Definition Language (DDL)
3. Data Manipulation Language (DML)
4. Data Query Language (DQL)

Data Manipulation Language (DML)

- Chargement de fichiers dans des tables
- `LOAD DATA [LOCAL] INPATH 'filepath' [OVERWRITE] INTO TABLE tablename [PARTITION (partcol1=val1, partcol2=val2 ...)]`
- `[LOCAL]` : optionnelle, indiquant que le fichier se trouve dans le système de fichier locale.
- `'filepath'` : indique le chemin du fichier.
- `[OVERWRITE]` : optionnelle, indiquant que le contenu de la table cible sera supprimé et remplacé par le fichier, si non, le fichier sera ajouté à la table.
- `LOAD DATA [LOCAL] INPATH 'filepath' INTO TABLE student`

Data Manipulation Language (DML)

- `[PARTITION (partcol1=val1, partcol2=val2 ...)]` : optionnelle, permettant d'insérer directement dans une partition.
- La table cible doit être créée avec la même colonne de partition.
- Il faut indiquer la colonne de partition ainsi que la valeur.
- Si la partition existe déjà, les données seront insérées dedans.
- Si la partition n'existe pas, elle sera créée et les données seront insérées dedans.

- ```
  LOAD DATA LOCAL INPATH 'filepath' INTO TABLE student
       PARTITION(gender='Male')
  ```
  ```

Data Manipulation Language (DML)

- Insertion à partir d'une requête dans des tables
- `INSERT {OVERWRITE | INTO} TABLE tablename1`

```
[PARTITION (partcol1=val1, partcol2=val2 ...) [IF NOT EXISTS]]
select_statement1 FROM from_statement;
```

- `INSERT OVERWRITE` : Remplace les données existantes.
- `INSERT INTO` : Ajoute les Nouvelles données sans toucher aux données existantes.

- `INSERT INTO TABLE studentMale SELECT * FROM student WHERE gender='Male'`

Data Manipulation Language (DML)

- `[PARTITION]` : optionnelle, supporte le partitionnement dynamique ou statique, permettant d'insérer directement dans une partition.
- Dans une partition dynamique, il suffit d'indiquer le ou les noms des colonnes de partition dans la clause PARTITION (sans valeur). Les valeurs des colonnes sont optionnelles. Si nous indiquons une valeur de colonne, la partition est appelée partition statique.
- Chaque colonne d'une partition dynamique doit avoir une colonne source de l'instruction select. Ainsi, la création de partition dynamique est déterminée par les valeurs des colonnes sources.

Data Manipulation Language (DML)

- Les colonnes de partition doivent être spécifiées dans l'instruction select dans le même ordre de la clause PARTITION.
- Les colonnes de partition doivent être les derniers dans l'instruction select.
- Par défaut, la partition est statique. Pour activer la partition dynamique, il faut exécuter : `SET hive.exec.dynamic.partition.mode=nonstrict;`

```
INSERT OVERWRITE TABLE studentPart PARTITION(gender)
      SELECT name, city, moy, gender  FROM student
```

Data Manipulation Language (DML)

- Dans une partition statique, les valeurs des colonnes dans la clause PRTITION sont obligatoires.
- Avec un partitionnement statique, il faut gérer manuellement chaque partition : créer la partition avec ALTER, puis insérer dans la partition.

- `ALTER TABLE studentPart ADD PARTITION (gender='Male');`

- ```
  INSERT OVERWRITE TABLE studentPart PARTITION(gender='Male')
      SELECT name, city, moy FROM student WHERE gender='Male';
  ```
  ```

Data Manipulation Language (DML)

- Dans la clause PARTITION, le nom de colonne de partition ainsi qu'une valeur spécifique sont indiqués.
- Dans l'instruction select, la colonne de partition n'est pas spécifiée.
- Dans la clause WHERE, la condition indique que seules les lignes qui correspondent à la valeur spécifique de la colonne de partition seront ramenées.
- Avec le partitionnement statique, le programmeur est le seul responsable d'assurer que les données seront stockées dans les partitions correctes.

Data Manipulation Language (DML)

- Insertion de valeurs saisies au clavier dans des tables
- `INSERT INTO TABLE tablename`

  `[PARTITION (partcol1[=val1], partcol2[=val2] ...)]`

  `VALUES values_row [, values_row ...]`
- Chaque ligne indiquée dans la clause VALUES sera insérée dans la table.
- Le nombre de valeurs doit être le même que celui de colonne de la table. Utiliser ***null*** pour insérer une valeur nulle.
- Le partitionnement dynamique est supporté.
- Exemple: table student sans partition

```
INSERT INTO TABLE students VALUES ('fred', 'NY', 14.28,'Male')
```

Data Manipulation Language (DML)

- La table student est avec partition (mode de partition dynamique)

```
INSERT INTO TABLE  students PARTITION (gender = ’Male')
VALUES ('smith', ’NY’,15);
INSERT INTO TABLE  students PARTITION (gender)
VALUES ('smith', ’NY’,15,’Male’);
```

- La table student est avec partition (mode de partition statique)

La partition pour gender = Male doit être créée avec ALTER, puis

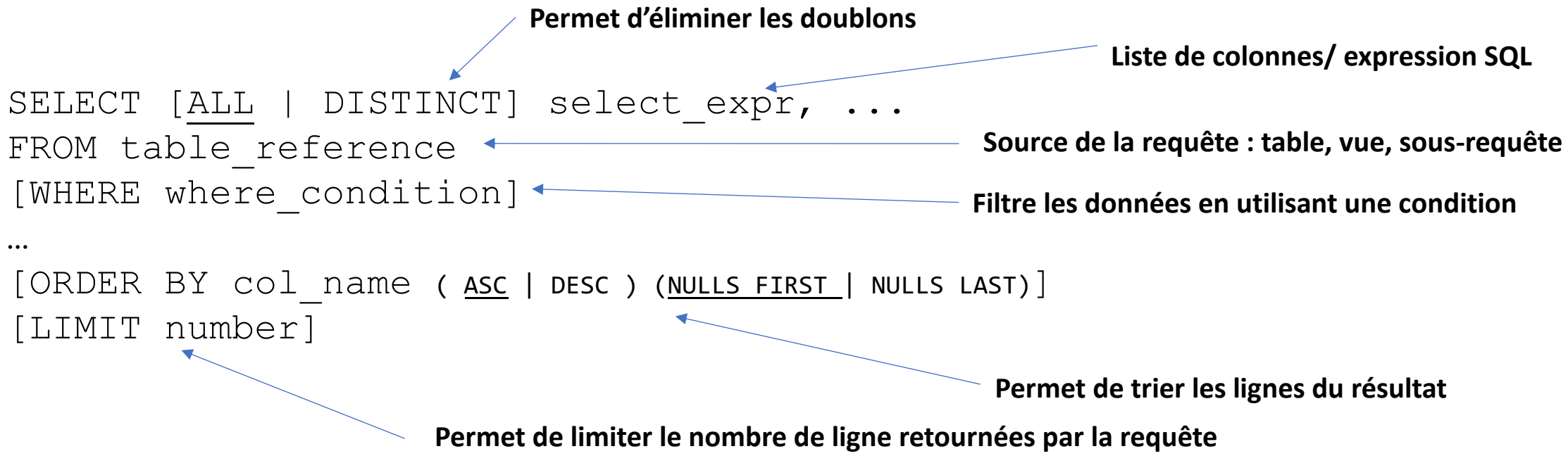
```
INSERT INTO TABLE  students PARTITION (gender = ’Male')
VALUES ('smith', ’NY’,15);
```

Partie 2
Hive: Distrubuted Data Warehouse

1. Aperçu et architecture
2. Data Definition Language (DDL)
3. Data Manipulation Language (DML)
4. Data Query Language (DQL)

Data Query Language (DQL)

- Le langage de requête Hive (HiveQL) est un langage de requête permettant à Hive de traiter et d'analyser des données structurées.
- L'instruction SELECT permet de récupérer les données d'une table.



Data Query Language (DQL)

```
SELECT * FROM student limit 2;
```

Name	Gender	Moy
ALLEN	Male	12.25
Emma	Female	10.5

Name	Gender	Moy
ALLEN	Male	12.25
Emma	Female	10.5
JONES	Male	14.66
ADAMS	Male	
Sophia	Female	10.5
MILLER	Male	9.5
Charlotte	Female	

```
SELECT distinct gender, moy FROM
student where gender = 'Female';
```

Gender	Moy
Female	10.5
Female	

```
SELECT distinct name, moy FROM
student order by moy;
```

Name	Moy
ADAMS	
Charlotte	
MILLER	9.5
Emma	10.5
Sophia	10.5
ALLEN	12.25
JONES	14.66

Data Query Language (DQL)

- Opérateurs relationnels :

```
=, !=, <, <=, >, >=, [not] between, [not] like, is[not] null
```

- Opérateurs arithmétiques : `*, /, %, +, -`
- Opérateurs logiques : `and, or, not, [not] in`

Si l'un des opérandes est null, le résultat sera null

```
SELECT * FROM student where
note1>note2;
```

Name	Note1	Note2
Sophia	13	10.5

```
SELECT * FROM student where
note1>=12 and name like '_D%';
```

Name	Note1	Note2
ADAMS	12	

Name	Note1	Note2
ALLEN	10	12.25
ADAMS	12	
Sophia	13	10.5
Charlotte	15	

Data Query Language (DQL)

- Fonctions arithmétiques :

round(DOUBLE a[, INT d]) valeur arrondie de A à d chiffres après virgule **round(16,8) -> 17**

floor(DOUBLE a) : le plus grand entier < a **floor(16,8) ->16**

ceil(DOUBLE a) : le plus petit entier > a **ceil(16,8) -> 17**

pow(DOUBLE a, DOUBLE p) : ap

sqrt(DOUBLE a)

abs(DOUBLE a)

greatest(v1, v2, ...) : la plus grande valeur de la liste, null si une valeur est nulle.

least(v1, v2, ...) : la plus petite valeur de la liste, null si une valeur est nulle.

Data Query Language (DQL)

Name	Gender	Moy
ALLEN	Male	12.25
Emma	Female	10.5
JONES	Male	14.66
ADAMS	Male	
Sophia	female	10.5
MILLER	Male	9.5
Charlotte	FEMALE	

- Fonctions de chaine de caractère :

concat(string A, string B...) : concatenation de B à la fin de A

length(string A) : retourne la longueur de la chaine

lower(string A) / **upper**(string A) : retourne A en miniscule / majuscule

ltrim(string A) / **rtrim**(string A) : enlève les espaces à gauche /à droite de A

initcap(string A) : met la première lettre de A en majuscule

```
SELECT concat(name,' is ',gender)
FROM student limit 2;
```

…
ALLEN is Male
Emma is Female

```
SELECT * FROM student
lower (gender)='female';
```

Name	Gender	Moy
Emma	Female	10.5
Sophia	Female	10.5
Charlotte	FEMALE	

Data Query Language (DQL)

Name	Note1	Note2
ALLEN	10	12.25
ADAMS	12	
Sophia	13	10.5
Charlotte	15	

- Fonctions conditionnelles :

`if(boolean testCondition, T valueTrue, T valueFalse)` : retourne valueTrue si la condition est vraie, valueFalse sinon

`nvl(value, T default_value)` : retourne default_value si value est null. Value sinon.

`nullif( a, b )` : retourne null si a=b, a sinon

```
SELECT note1,if(note1>10,'> moy','< moy')
as result FROM student;
```

Note1	result
10	> moy
12	> Moy
13	> Moy
15	> moy

```
SELECT name,note1 + note2 as somme
FROM student;
```

Name	somme
ALLEN	22.25
ADAMS	Null
Sophia	23.5
Charlotte	null

```
SELECT name,note1 + nvl(note2,0)
as somme FROM student;
```

Name	somme
ALLEN	22.25
ADAMS	12
Sophia	23.5
Charlotte	15

Data Query Language (DQL)

Name	Note1	Note2
ALLEN	10	12.25
ADAMS	12	
Sophia	13	10.5
Charlotte	15	

- Fonctions d'agrégation :

`count(*)` : retourne le nombre de lignes

`count([DISTINCT] expr)` : retourne le nombre de valeurs [distinctes] pour lesquelles expr est non null

`sum(col), sum([DISTINCT] col)` : retourne la somme des valeurs [distinctes] non null de col

`avg([DISTINCT] col)` : retourne la moyenne des valeurs [distinctes] non null de col

`min(col) / max(col)` : retourne le min / max des valeurs de col

```
SELECT count(note2) FROM student;
```


```
SELECT sum(note2) FROM student;
```


```
SELECT name,sum(note2) FROM student;
```

Data Query Language (DQL)

Name	Note1	Note2
ALLEN	10	12.25
ADAMS	12	
Sophia	13	10.5
Charlotte	15	

- Casting

`cast(expr as <type>)` : Convert expr en <type>

- Maskage de donnée

`mask(string str[, string upper[, string lower[, string number]]])` : retourne une version maskée de str. Caractère majuscule en X, Caractères miniscule en x, nombre en n.

```
SELECT mask(name), mask(cast(note1
as string)) FROM student;
```

...	...
XXXXX	nn
XXXXX	nn
Xxxxxx	nn
Xxxxxxxxx	nn

```
SELECT mask(name,"B","m"),
mask(cast(note1 as
string),"B","m","X") FROM student;
```

...	...
BBBBB	XX
BBBBB	XX
Mmmmmm	XX
Mmmmmmmmm	XX

Data Query Language (DQL)

Name	birthday
ALLEN	2000-10-23
ADAMS	2001-5-12
Sophia	1999-12-13
Charlotte	2001-3-15

- Fonctions de date :

`year(string date) / month(string date) / day(string date) :` retourne l'année / mois / jour d'une date.

`weekofyear(string date) :` retourne la semaine de l'année.

`datediff(date1, date2) / months_between(date1, date2) :` retourne le nombre de jours / mois entre date1 et date2

`date_add(date, tinyint/smallint/int days) / add_months(date, int months) :` ajoute days / months à date

`date_sub(date, tinyint/smallint/int days) :` soustrait days de date

```
SELECT name, year(birthday) FROM
student;
```

Name	…
ALLEN	2000
ADAMS	2001
Sophia	1999
Charlotte	2001

```
SELECT name,
cast(months_between(current_date,birthday)as int)
FROM student;
```

Name	…
ALLEN	251
ADAMS	245
Sophia	262
Charlotte	247

Data Query Language (DQL)

- Fonction de groupe :

`Group by` `colonne :` permet de regrouper (classifier, diviser) les données d'une table en groupes. colonne est le critère de classification.

`Having` `condition :` permet de définir une condition sur les groupes

Remarques :

- Les fonctions d'agrégation ne peuvent pas être utilisées dans la clause WHERE.
- Avec la clause GROUP BY, on ne peut mettre dans la clause SELECT qu'une caractéristique du groupe : critère de groupement et/ou fonction de groupe

Data Query Language (DQL)

```
SELECT gender, count(*) Number
FROM student group by gender;
```

gender	Number
Male	4
Female	3

Name	Gender	Moy
ALLEN	Male	12.25
Emma	Female	10.5
JONES	Male	14.66
ADAMS	Male	
Sophia	Female	10.5
MILLER	Male	9
Charlotte	Female	

```
SELECT gender, avg(moy) MoyParGender
FROM student group by gender;
```

gender	MoyParGender
Male	11,97
Female	10,5

```
SELECT Name, gender, avg(moy) MoyParGender FROM student group by gender;
```

Gender dont la moyenne est > 12.

```
SELECT gender, avg(moy) MoyMaxGender FROM student where avg(moy)>12 group by gender ;
```

```
SELECT gender, avg(moy) MoyMaxGender
FROM student
group by gender
having MoyMaxGender > 12;
```

gender	MoyMaxGender
Male	14,66

Data Query Language (DQL)

- L'opérateur ensembliste UNION

`select_statement` **`UNION`** `[ALL | DISTINCT] select_statement` : Permet d'avoir les résultats de la première requête suivis de ceux de la deuxième requête. Par défaut les doublons sont éliminés.

- Les deux SELECT doivent avoir le même nombre de colonnes. Ces colonnes doivent être de types compatibles.
- Les titres des colonnes résultats sont ceux de la première requête.
- ORDER BY et LIMIT peuvent être appliquées au résultat final

Les Male et les Female dont la moyenne n'est pas définie

```
SELECT * FROM student
where gender = 'Male' and moy is null
Union
SELECT * FROM student
where gender = 'Female' and moy is null
```


Name	Gender	Moy
ADAMS	Male	NULL
Charlotte	Female	NULL

Data Query Language (DQL)

- Jointure : Elle permet de combiner des enregistrements issus à partir de deux ou plusieurs tables en vue de retrouver des données associées.
- Jointure interne : join | inner join
- Les lignes qui n'ont pas de correspondance ne sont pas ramenées.

```
SELECT name, libelle FROM student join
department on dep=id ;

SELECT name, libelle
FROM student, department where dep=id ;
```

Name	Libelle
ALLEN	MID
Emma	MID
JONES	Telecom
ADAMS	Telecom
Sophia	MID
Charlotte	Telecom

Name	Gender	Moy	Dep
ALLEN	Male	12.25	1
Emma	Female	10.5	1
JONES	Male	14.66	2
ADAMS	Male		2
Sophia	Female	10.5	1
MILLER	Male	9	
Charlotte	Female		2

id	Libelle
1	MID
2	Telecom
3	Finance

Data Query Language (DQL)

- Jointure externe :Les lignes qui n'ont pas de correspondance sont ramenées
- full outer join

```
SELECT name, libelle FROM student full outer join
department on dep=id ;
```

Name	Libelle
ALLEN	MID
Emma	MID
JONES	Telecom
ADAMS	Telecom
Sophia	MID
MILLER	NULL
Charlotte	Telecom
NULL	Finance

- left outer join

```
SELECT name, libelle FROM student left outer join
department on dep=id ;
```

Name	Libelle
ALLEN	MID
Emma	MID
JONES	Telecom
ADAMS	Telecom
Sophia	MID
MILLER	NULL
Charlotte	Telecom

- right outer join

```
SELECT name, libelle FROM student right outer
join department on dep=id ;
```

Name	Libelle
ALLEN	MID
Emma	MID
JONES	Telecom
ADAMS	Telecom
Sophia	MID
Charlotte	Telecom
NULL	Finance