DS501 Big Data Technologies – Pig, Hive

By: John

BITTIGER

The Lifelong Learning Platform of Silicon Valley

版权声明

所有太阁官方网站以及在第三方平台课程中所产生的课程内容，如文本，图形，徽标，按钮图标，图像，音频剪辑，视频剪辑，直播流，数字下载，数据编辑和软件均属于太阁所有并受版权法保护。

对于任何尝试散播或转售BitTiger的所属资料的行为，太阁将采取适当的法律行动。

Copyright Policy

All content included on the Site or third-party platforms as part of the class, such as text, graphics, logos, button icons, images, audio clips, video clips, live streams, digital downloads, data compilations, and software, is the property of BitTiger or its content suppliers and protected by copyright laws.

Any attempt to redistribute or resell BitTiger content will result in the appropriate legal action being taken.

We thank you in advance for respecting our copyrighted content.

For more info:
see https://www.bittiger.io/termsofuse
and https://www.bittiger.io/termsofservice

Outline

Data processing

Pig

Hive

Data Processing

Data Processing

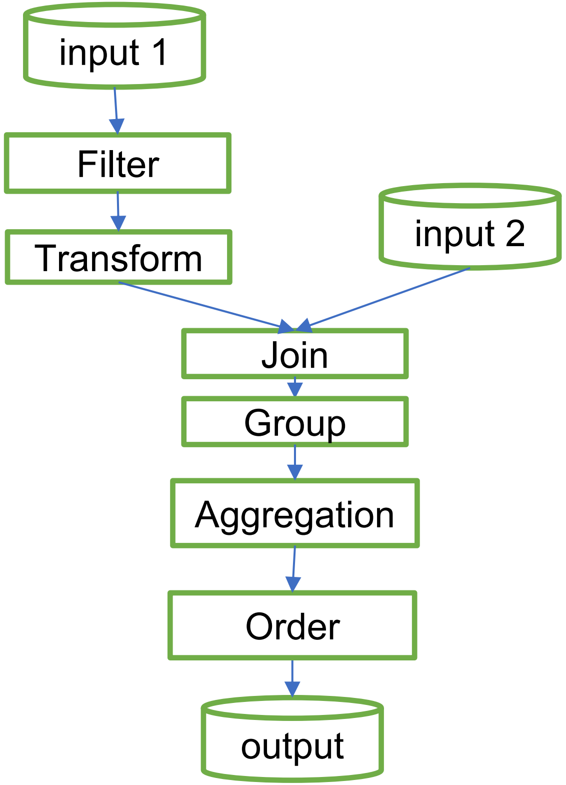
- Major use case of Hadoop
  - In Yahoo!, 90% of Hadoop job are Pig job
  - Facebook uses mainly Hive for data analysis
  - Hand write a data processing program in Hadoop is hard
- Spark & Flink
  - Data processing is the only use case

Character of Data Processing

- Very few operations
  - Filter, transform (foreach, map)
  - Aggregation
  - Join
  - Load, store
- UDF

Common Operator in Data Processing

- Filter
- Transform (Foreach)
- Join
- Group
- Aggregation
- Order

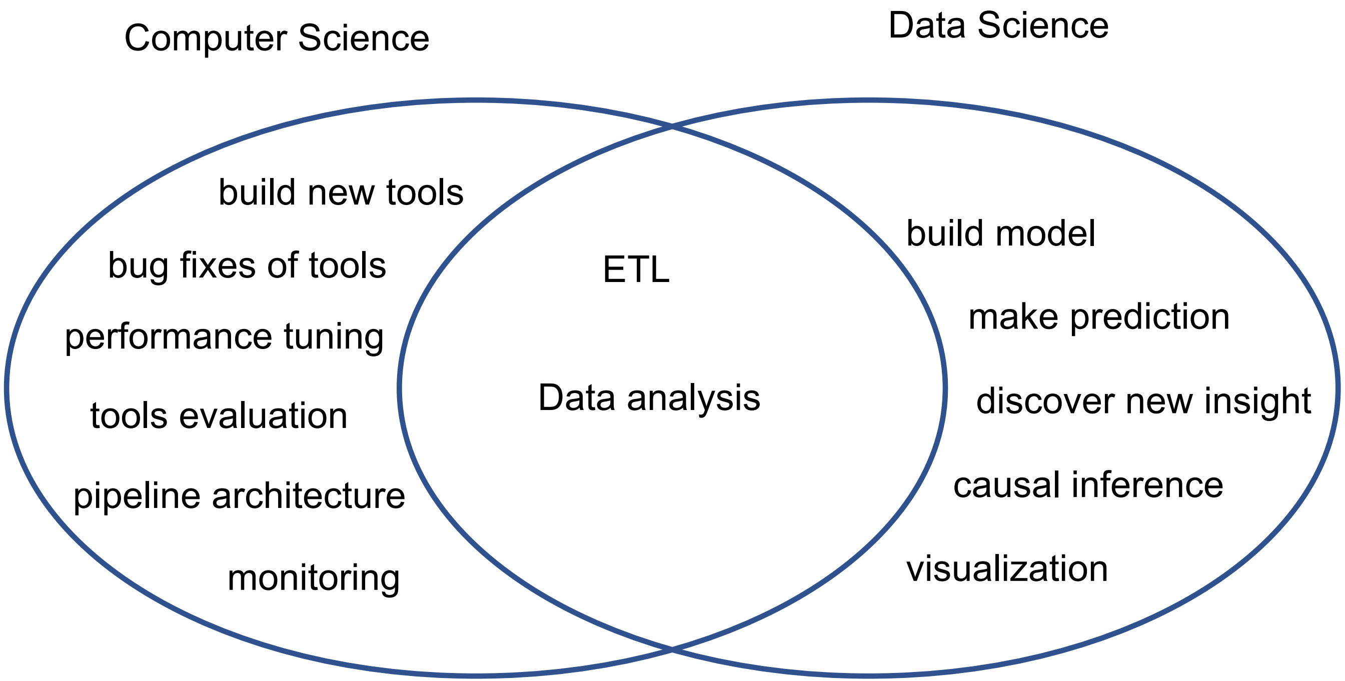


Goal of Data Processing

BITTIGER

- Statistics
  - Hot spot
  - Trend
  - Correlation
- Visualization
- Build prediction model

Data Engineer vs Data Science



Computer Science

Data Science

build new tools

bug fixes of tools

performance tuning

tools evaluation

pipeline architecture

monitoring

ETL

Data analysis

build model

make prediction

discover new insight

causal inference

visualization

Pig

Outline

What is Pig

Pig Data Types

Pig Syntax (Pig Latin)

What is Pig

BITTIGER

- Pig Latin, a high level data processing language
- An engine that executes Pig Latin locally or on a Hadoop cluster
    - MapReduce
    - Tez
    - Spark

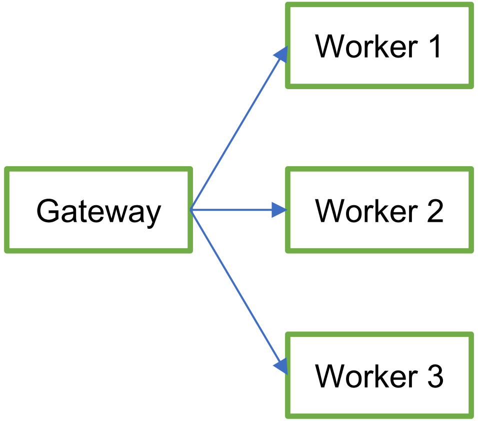
What is Pig

- Query : Get the list of web pages visited by users whose age is between 20 and 29 years

```
USERS = load ‘users’ as (uid, age);

USERS_20s = filter USERS by  age >= 20 and age <= 29;

PVs = load ‘pages’ as (url, uid, timestamp);

PVs_u20s  = join USERS_20s by uid, PVs by uid
```

Run Pig with Different Engine

- MR
  - pig -x mr (or default)
- Spark
  - pig –x spark
- Tez
  - pig –x tez



Interactive Mode and Script Mode

- Run Pig script
  - pig script.pig
- Run interactively with Grunt shell
  - grunt>

Prime Datatype

- Simple Datatype
  - int, long, float, double, boolean
  - chararray
  - bytearray
  - datetime
  - biginteger/bigdecimal

Complex Type

- Map
- Tuple
- Bag

Map

BITTIGER

- Java HashMap = Python dictionary
- chararray->object
- Map constant: ['name'#'bob', 'age'#55]
- Key reference: m#'name'

Tuple

- List of items = Python list/tuple
- Tuple constant: ('bob', 55)

Bag

- Unordered collection of tuples, similar to Python list
- {('bob', 55), ('sally', 52), ('john', 25)}
- Access a particular tuple is not possible, must iterate
- Slice a bag: b.$0: {('bob'), ('sally'), ('john')}
- Will spill to disk

Pig Script

```
a = load 'studenttab10k' using PigStorage() as (name:chararray, age:int,
gpa:double);

a1 = filter a by age > 18;

a2 = foreach a1 generate name, ROUND(gpa) as gpa;

b = load 'votertab10k' using PigStorage() as (name:chararray, age:int,
registration:chararray, contributions:double);

c = join a2 by name, b by name;

d = group c by registration;

e = foreach d generate group, AVG(c.gpa) as gpa;

f = order e by gpa desc;

dump f;
```

Load

- a = load 'studenttab10k' using PigStorage() as (name:chararray, gender:chararray, age:int, gpa:double);
  - 'studenttab10k': file location on HDFS or local
  - PigStorage: LoadFunc
  - as (name:chararray, age:int, gpa:double): optional schema
- Unknown schema is also acceptable, use position to refer field: $0, $1…
- Some LoadFunc get schema automatically from data: AvroStorage, OrcStorage

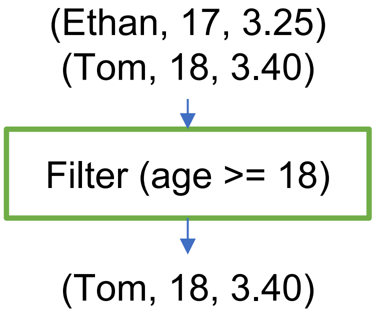
Store

- store a into 'output' using PigStorage();
  - PigStorage: StoreFunc
  - 'output' : file location on HDFS or local
  - Will infer schema from Pig Script

Filter

BITTIGER

- a1 = filter a by age >= 18;
- a1 = filter a by age >= 18 and gpa > 3;
- a1 = filter a by IsAdult(age); -- filter by UDF



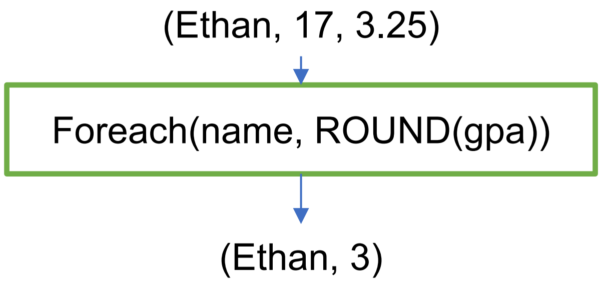
(Ethan, 17, 3.25)
(Tom, 18, 3.40)

Filter (age >= 18)

(Tom, 18, 3.40)

Foreach

- a2 = foreach a1 generate name, ROUND(gpa) as gpa; --UDF
- a2 = foreach a1 generate name, gpa>3?1:0; --bincond
- e = foreach d generate group, AVG(c.gpa) as gpa; --aggregation



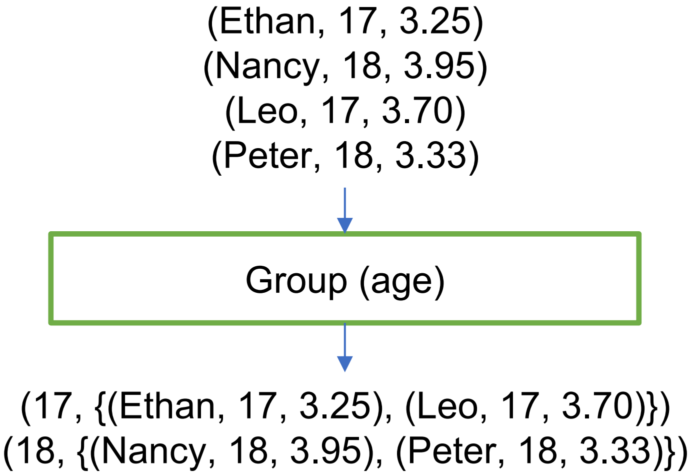
(Ethan, 17, 3.25)

Foreach(name, ROUND(gpa))

(Ethan, 3)

Group

- d = group c by age;
- Result: key + bag



(Ethan, 17, 3.25)
(Nancy, 18, 3.95)
(Leo, 17, 3.70)
(Peter, 18, 3.33)

Group (age)

(17, {(Ethan, 17, 3.25), (Leo, 17, 3.70)})
(18, {(Nancy, 18, 3.95), (Peter, 18, 3.33)})

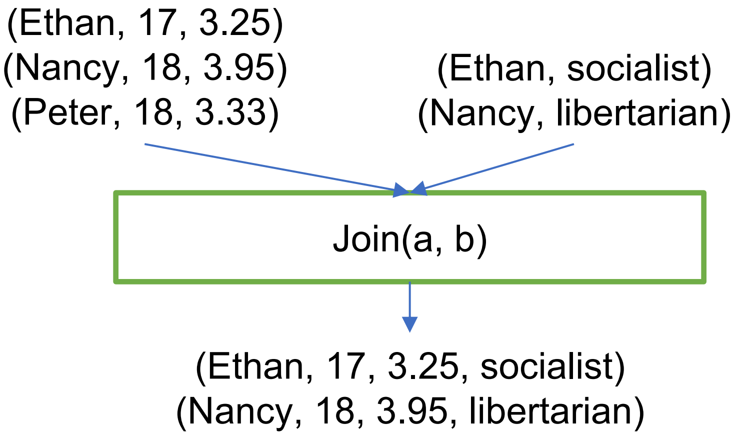
Group all

- d = group c all;
- SQL select COUNT(*) equivalent in Pig
- Only use 1 reduce

```
a = load 'studenttab10k' as
(name:chararray, age:int, gpa:double);
b = group a all;
c = foreach b generate COUNT(a);
```

Join

- c = join a2 by name, b by name;



(Ethan, 17, 3.25)
(Nancy, 18, 3.95)
(Peter, 18, 3.33)

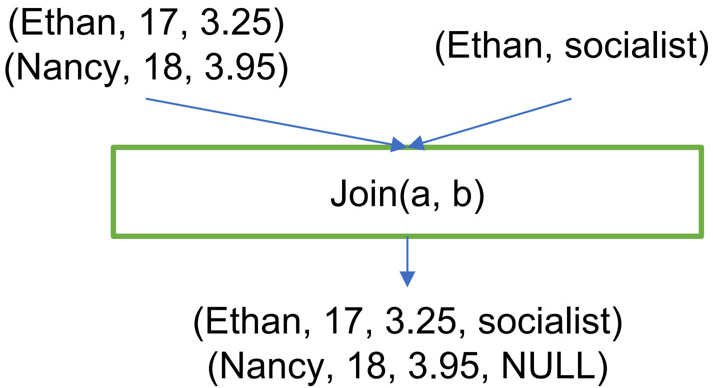
(Ethan, socialist)
(Nancy, libertarian)

Join(a, b)

(Ethan, 17, 3.25, socialist)
(Nancy, 18, 3.95, libertarian)

Outer Join

- c = join a2 by name left outer, b by name;



(Ethan, 17, 3.25)
(Nancy, 18, 3.95)

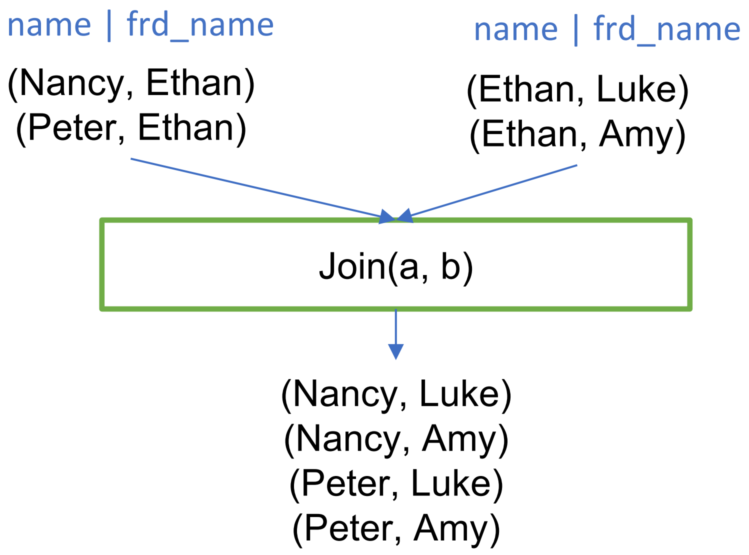
(Ethan, socialist)

Join(a, b)

(Ethan, 17, 3.25, socialist)
(Nancy, 18, 3.95, NULL)

Join

- c = join a by frd_name, b by name;



name | frd_name

(Nancy, Ethan)
(Peter, Ethan)

name | frd_name

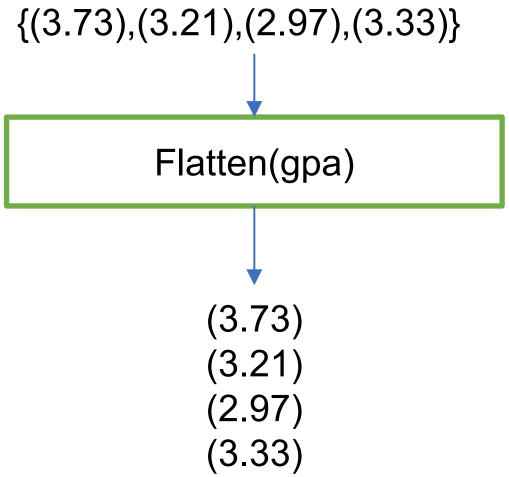
(Ethan, Luke)
(Ethan, Amy)

Join(a, b)

(Nancy, Luke)
(Nancy, Amy)
(Peter, Luke)
(Peter, Amy)

Foreach ... Flatten

- c = foreach b generate flatten(a.gpa)
- A bag of N tuples => N rows



{(3.73),(3.21),(2.97),(3.33)}

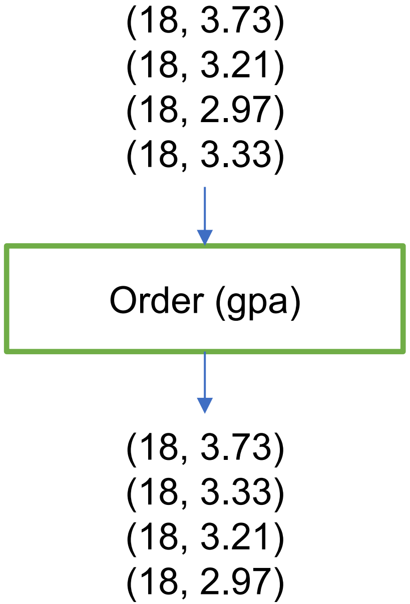
Flatten(gpa)

(3.73)
(3.21)
(2.97)
(3.33)

Order

- f = order e by gpa desc;

  (asc/desc)



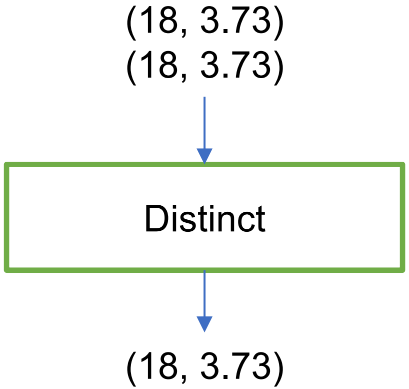
(18, 3.73)
(18, 3.21)
(18, 2.97)
(18, 3.33)

Order (gpa)

(18, 3.73)
(18, 3.33)
(18, 3.21)
(18, 2.97)

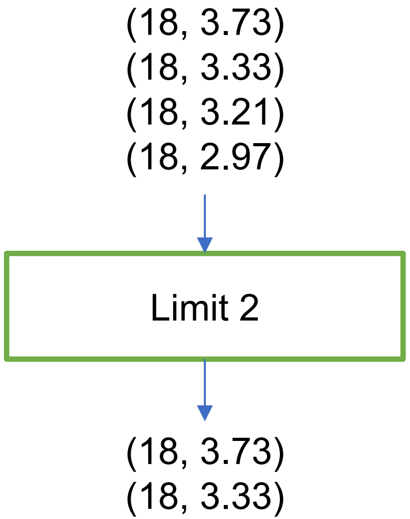
Distinct

- f = distinct e;
- You can only distinct the whole tuple



Limit

- g = limit e 2;
- After order, top query
- Otherwise, pick any 2



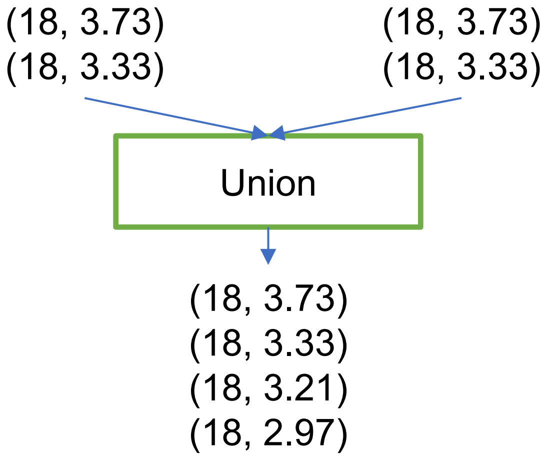
(18, 3.73)
(18, 3.33)
(18, 3.21)
(18, 2.97)

Limit 2

(18, 3.73)
(18, 3.33)

Union

- c = union a, b;



(18, 3.73)
(18, 3.33)

(18, 3.73)
(18, 3.33)

Union

(18, 3.73)
(18, 3.33)
(18, 3.21)
(18, 2.97)

Pig Script

```
a = load 'studenttab10k' using PigStorage() as (name:chararray,
gender:chararray, age:int, gpa:double);

a1 = filter a by age > 18;

a2 = foreach a1 generate name, ROUND(gpa) as gpa;

b = load 'votertab10k' using PigStorage() as (name:chararray, age:int,
registration:chararray, contributions:double);

c = join a2 by name, b by name;

d = group c by registration;

e = foreach d generate group, AVG(c.gpa) as gpa;

f = order e by gpa desc;

dump f;
```

Hive

Outline

What is Hive

HiveQL

What is Hive

- SQL engine on Hadoop: HiveQL
- Multi-engine (HiveQL convert queries to run the following jobs)
    - MapReduce
    - Tez
    - Spark

Why SQL

- Most data analyst know SQL
- Standard language, integrate with existing BI tool
  - Tableau
  - Pentaho
  - Qlik

History

BITTIGER

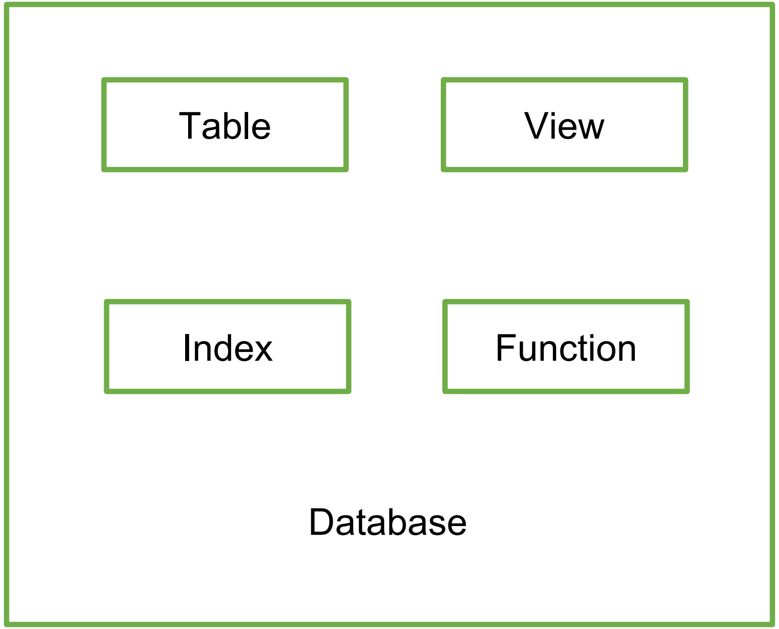
- 2006-2007 Internal development in Facebook
- 2008 Hadoop sub-project
- 2010 Graduate to Apache TLP
- 2011-2012 Interim
- 2013 Hortonworks become the new backbone

Hive Features

- Indexing to provide acceleration
- Storage types: plain text, RCFile, HBase, ORC, and others.
- Metadata storage in a relational database management system
- Operating on compressed data stored into the Hadoop ecosystem
- UDFs: Built-in UDFs to manipulate dates, strings, and other data-mining tools. Hive supports extending the UDF set to handle more use-cases.
- SQL-like queries (HiveQL): implicitly converted into MapReduce or Tez, or Spark jobs. While based on SQL, HiveQL does not strictly follow the full SQL-92 standard.

Hive Object

- Database
- Table
- View
- Index
- Function



Database

- Location on HDFS
  - /user/hive/warehouse/$database
  - Configurable

```
create database cs502; -- create a database

use cs502; -- Change current database

drop database cs502 cascade; -- drop a database
```

Create Table

```
CREATE TABLE student(name string, age int, gpa double)
PARTITIONED BY(gender string)
CLUSTERED BY(name) INTO 4 BUCKETS;
```

```
student/gender=M/000000_0
                /000000_1
                /000000_2
                /000000_3
       /gender=F/000000_0
                /000000_1
                /000000_2
                /000000_3
```

Table Partition Types

- Partition
  - Range partition (The data is distributed based on a range of values)
- Bucket
  - Hash partition (An internal hash algorithm is applied to the partitioning key to determine the partition.)

Why Partition?

- **Decreases costs** by storing data in the most appropriate manner.
- **Increases performance** by only working on the data that is relevant.
- **Improves availability** through individual partition manageability.

Schema

BITTIGER

- Simple type
  - tinyint, smallint, int, bigint, float, double, decimal
  - timestamp, date, interval
  - string, varchar, char
  - boolean, binary
- Complex type
  - array, map, struct

Managed Table vs External Table

- Managed table
  - Data under /usr/hive/warehouse
  - Drop table also drop data
- External table
  - Data in external location
  - Drop table keep data

```
CREATE EXTERNAL TABLE student_ext(name string,
age int, gpa double)
LOCATION '/data/student';
```

Alter Table

- Add/drop partition
- Change table properties
- Alter column name/type

```
ALTER TABLE student ADD
PARTITION(gender='F', state='CA');
```

Load Data into Table

- Load data into table either from local fs or HDFS

```
LOAD DATA LOCAL INPATH 'studenttab10k' INTO TABLE student_src;

LOAD DATA INPATH 'studenttab10k' INTO TABLE student_src;

LOAD DATA LOCAL INPATH 'studenttab10k' INTO TABLE student
PARTITION(gender='M', state='CA');
```

View

- Virtual table
- Not materialized

```
create view male_student as select name,
age, gpa from student where gender = 'M';


select COUNT(*) from male_student;
```

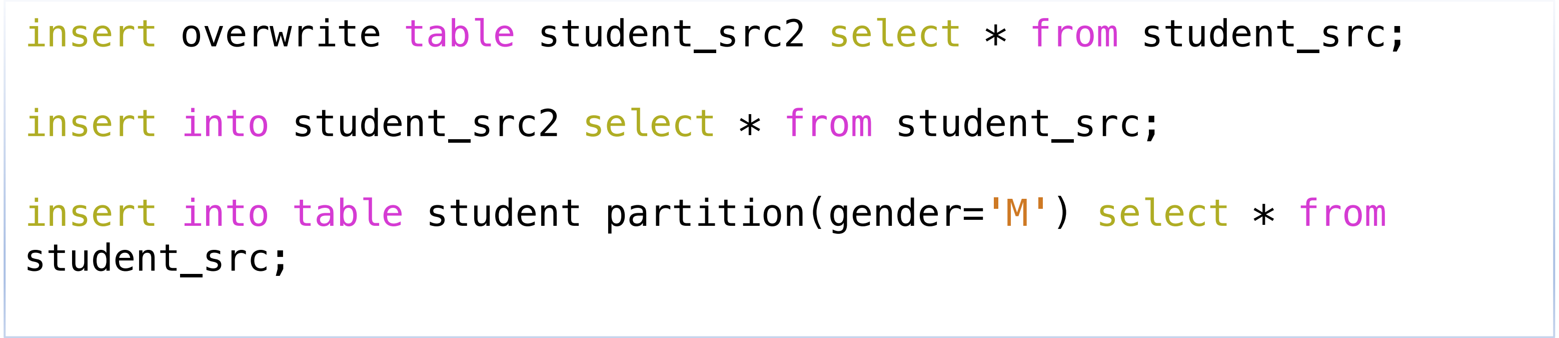
Function

- Add jar (=Pig register)
- Create temporary function (=Pig define)

```
add jar myudf.jar;

create temporary function my_lower as 'com.example.hive.udf.Lower';
```

Insert

- Insert data into table/partition



```
insert overwrite table student_src2 select * from student_src;

insert into student_src2 select * from student_src;

insert into table student partition(gender='M') select * from
student_src;
```

Select

```
SELECT name, age, gpa FROM student WHERE gender = 'M';

SELECT name, AVG(gpa) avg_gpa FROM student GROUP BY name
HAVING avg_gpa > 2.8 ORDER BY avg_gpa DESC;

SELECT * FROM student ORDER BY gpa; --single reducer
```

Join

```
SELECT s.name, contributions FROM student s JOIN voter v ON
s.name = v.name;
```

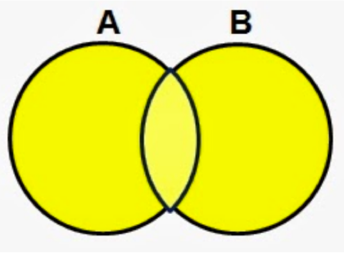
```
SELECT s.name, contributions FROM student s LEFT JOIN voter
v ON s.name = v.name;
```

Join Type

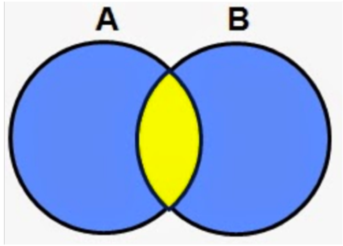
- Common join
- Map side join
- Skewed join
- Bucket map join
- Sorted merge bucket join

Set Operation

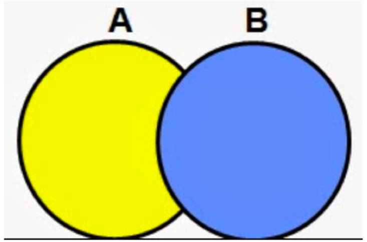
- Union, Union All
- Intersect (Hive 2)
- Except (Hive 2)



A Union All B



A Intersect B



A Except B

Subquery

```
SELECT s1.name, AVG(s1.gpa) FROM (SELECT name, age, gpa from
student where gpa>3) s1 group by s1.name;

SELECT s1.name, s1.gpa FROM (SELECT name, age, gpa FROM student
WHERE gpa>3) s1 JOIN (SELECT DISTINCT name FROM student WHERE
gpa < 3.5) s2 ON s1.name = s2.name;

SELECT name FROM student s WHERE age in (select age from
voter);
```

Correlated subquery:

```
SELECT name FROM student s WHERE EXISTS (SELECT * FROM voter v
WHERE v.name=s.name AND v.contributions > 100); --very slow
```

Update and Delete

```
UPDATE student SET name = null WHERE gpa <= 1.0;

DELETE FROM student WHERE gpa <= 1,0;
```