



UNIVERSIDAD  
DE GRANADA



# Big Data I

Máster Universitario en Ciencia de Datos e Ingeniería de Computadores

Consulta estructurada sobre datos no estructurados  
masivos: Apache® Impala



I. J. Blanco

Departamento de Ciencias de la  
Computación e Inteligencia Artificial  
<http://decsai.ugr.es>

- Infraestructura para *DataWarehousing* basada en Hadoop.
- Permite:
  - extraer resúmenes de datos,
  - consultas *ad hoc* y
  - analizar grandes volúmenes de datos
- Es útil para grandes volúmenes de datos que no cambian.



- Es un lenguaje de consulta sobre flujos de datos.
- Permite al usuario especificar **qué** desea obtener del flujo de datos (lenguaje de 4ª generación) **y no** necesita decir **cómo**.
- Útil para ETL y para usuarios habituales de SQL o herramientas de *Business Intelligence*.
- *filtrado, agregación, cálculo u ordenación* son cosa del motor Impala.

- Hadoop: sistema de proceso por lotes gasta tiempo en organizar el trabajo.
- Las consultas en Impala sobre conjuntos de datos pequeños (pocos  $10^2$  MB) son muy lentas.
- No es útil para OLTP.
- No sirve para consultas en tiempo real.
- No es nada eficiente para realizar *updates* a nivel de fila (tupla).



```
package org.myorg;

import java.io.IOException;
import java.util.*;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class WordCount {

    public static class Map extends Mapper<LongWritable, Text, Text, IntWritable> {
        private final static IntWritable one = new IntWritable(1);
        private Text word = new Text();

        public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException
        {
            String line = value.toString();
            StringTokenizer tokenizer = new StringTokenizer(line);
            while (tokenizer.hasMoreTokens()) {
                word.set(tokenizer.nextToken());
                context.write(word, one);
            }
        }
    }
}
```



```
public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable> {

    public void reduce(Text key, Iterable<IntWritable> values, Context context)
        throws IOException, InterruptedException {
        int sum = 0;
        for (IntWritable val : values) {
            sum += val.get();
        }
        context.write(key, new IntWritable(sum));
    }
}

public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();

    Job job = new Job(conf, "wordcount");

    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);

    job.setMapperClass(Map.class);
    job.setReducerClass(Reduce.class);

    job.setInputFormatClass(TextInputFormat.class);
    job.setOutputFormatClass(TextOutputFormat.class);

    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

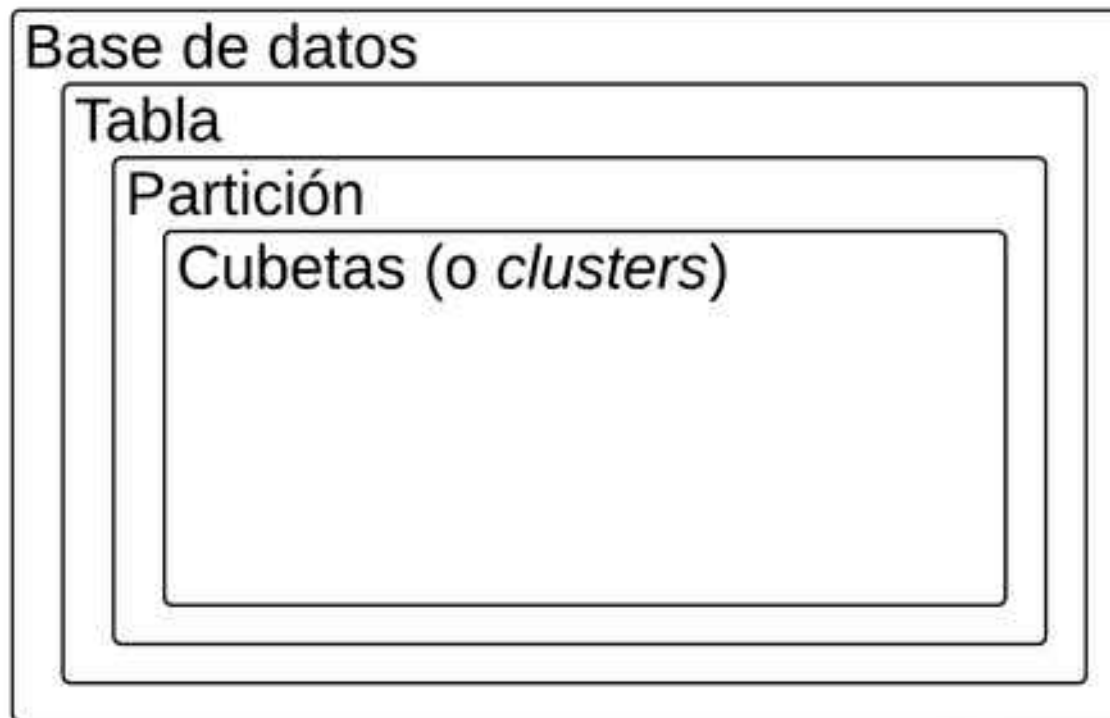
    job.waitForCompletion(true);
}
```



```
CREATE TABLE docs (line STRING);

LOAD DATA INPATH 'docs' OVERWRITE INTO TABLE
docs;

CREATE TABLE word_counts AS
  SELECT word, count(1) AS count FROM
    (SELECT explode(split(line, '\s')) AS
      word FROM docs) w
  GROUP BY word
  ORDER BY word;
```





## Base de Datos:

- Constituye un *namespace* (el conjunto de datos para un problema específico)
- Se permiten los mismos nombres de tablas en distintas bases de datos

## Base de Datos

### Tabla:

- Unidades de datos homogéneas que comparten esquema (estructura)

## Base de Datos

### Tabla

#### Partición:

- Se pueden usar para especificar dónde almacenar los datos en base a valores de clave
- Permiten hacer división lógica de los datos en base a valores de clave (identificación de filas)

## Base de Datos

### Tabla

### Partición

#### Cubeta (o *cluster*):

- Se pueden dividir los datos en función de el valor *hash* de una columna dada, aparte de la de la partición.

- Tablas gestionadas o internas:
  - se mueven dentro del almacenamiento de Impala,
  - pueden particionarse.
- Tablas externas:
  - se usan desde fuera del almacenamiento, permitiendo a otras aplicaciones acceder a ellas,
  - pueden estar particionadas.



```
CREATE TABLE [IF NOT EXISTS]
[db_name.]table_name

[(col_name data_type [COMMENT
'col_comment'], ...)]

[COMMENT 'table_comment']

[PARTITIONED BY (col_name data_type
[COMMENT 'col_comment'], ...)]

[ [ROW FORMAT row_format]
[STORED AS file_format] ]
[LOCATION 'hdfs_path']
```

- TINYINT (1 B)
- SMALLINT (2B)
- INT (4 B)
- BIGINT (8 B)
- BOOLEAN
- FLOAT
- DOUBLE
- STRING
- TIMESTAMP
- BINARY



- **TINYINT**: -128 .. 127 (no UNSIGNED)
- **SMALLINT**: -32768 .. 32767 (no UNSIGNED)
- **INT**: -2147483648 .. 2147483647
- **BIGINT**: -9223372036854775808 .. 9223372036854775807 (no UNSIGNED)
- **BOOLEAN**
- **FLOAT**: 1.40129846432481707e-45 .. 3.40282346638528860e+38 (positivos y negativos)
- **DOUBLE**: 4.94065645841246544e-324 .. 1.79769313486231570e+308 (positivos y negativos)
- **STRING**: hasta 32,767 bytes
- **TIMESTAMP**: desde el 1-1-1400 hasta el 31-12-9999



## **DELIMITED**

**[FIELDS TERMINATED BY 'char']**

**[ESCAPED BY 'char']]**

**[LINES TERMINATED BY 'char']**

```

CREATE 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 [COMMENT
'col_comment'], ...)]
[ [ROW FORMAT row_format]
[STORED AS file_format] ]
[LOCATION 'hdfs_path']

```

- Creación mediante:

```
CREATE EXTERNAL TABLE grados (<column
description>, ...)
ROW FORMAT DELIMITED FIELDS TERMINATED
BY ',' LINES TERMINATED BY '\n'
LOCATION
'/user/stark/datasets/grado.csv';
```



- **STRUCT:**

STRUCT<calle:STRING,numero:TINYINT,  
puerta:STRING,ciudad:STRING,cp:STRING>


- **MAP:**

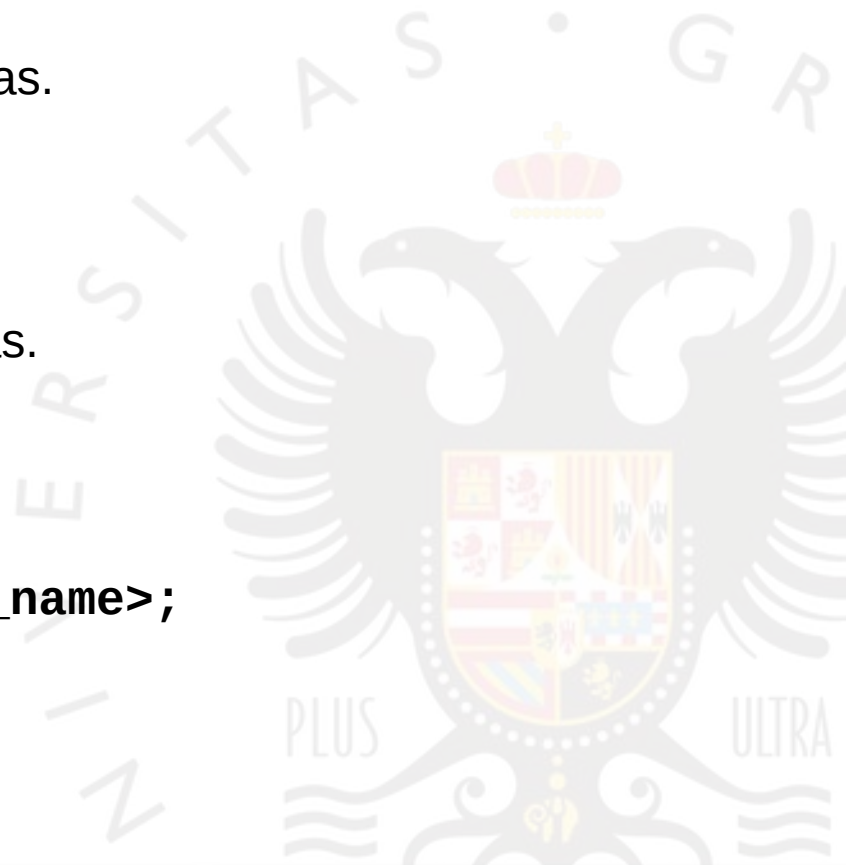
MAP<STRING,FLOAT>

- **ARRAY:**

ARRAY<STRING>



- Modificación de tablas (**ALTER TABLE**):
  - para renombrar tablas,
  - para re-ubicar tablas y
  - para particionar tablas no particionadas.
- Borrado de tablas (**DROP TABLE**): 
  - borra los metadatos de la tabla
  - pero no borra datos de tablas externas.
- Ver la estructura de las tablas:
  - **SHOW TABLES;**
  - **DESCRIBE [FORMATTED] <table\_name>;**



- Introducir registros en una tabla *interna*:
  - **INSERT [OVERWRITE | INTO]**  
**<table\_name> VALUES ...**
  - **INSERT [OVERWRITE | INTO]**  
**<table\_name> SELECT ...**



- Inicia la máquina virtual.
- Hay dos cuentas relevantes en el sistema operativo:
  - Username: `root`  
Para acceder, abre una ventana de terminal y ejecuta la orden “`sudo bash`”
  - Username: `impala`  
Para acceder, abre una ventana de terminal y ejecuta la orden “`sudo bash`” seguida de “`su - impala`”

El comando `whoami` te permitirá saber qué usuario está ejecutando la shell en cada momento.

- Abre una terminal con el usuario `impala`.
- Comprueba que el fichero de datos `BX-BooksCorrected.txt` está disponible y accesible:

```
ls /var/tmp/materialImpala
```

- Si no está disponible, crea el directorio con:

```
mkdir /var/tmp/materialImpala
```

y usa el navegador para descargar el fichero dentro del mismo pulsando el enlace:

<https://sl.ugr.es/matImpala2>



- Crea el directorio para los ficheros de ingesta de datos;

```
hdfs dfs -mkdir /user/impala/input
```

- Cargar los datos del fichero:

```
hdfs dfs -put /var/tmp/materialImpala/BX-BooksCorrected.txt /user/impala/input
```

- Comprobar que el fichero se ha cargado en HDFS:

```
hdfs dfs -ls /user/impala/input
```

- Acceder a Impala:

```
impala-shell
```

CREATE DATABASE IF NOT EXISTS `<dbname>`;

- Hay que tener acceso de escritura al directorio en el que se creará la base de datos.
- Como el HDFS está compartido con otras partes del curso, crearemos la DB en un directorio específico:

```
CREATE DATABASE IF NOT EXISTS <dbname>
LOCATION '/user/impala/impalastore.db';
```

- SHOW DATABASES;
- DESCRIBE DATABASE `<dbname>`;
- USE `<dbname>`;



- Crear una tabla:

```
CREATE TABLE IF NOT EXISTS BookData (ISBN STRING,  
BookTitle STRING, BookAuthor STRING,  
YearOfPublication INT, Publisher STRING) ROW FORMAT  
DELIMITED FIELDS TERMINATED BY '\\;' STORED AS  
TEXTFILE;
```

- Comprobar que la tabla se ha creado correctamente:

```
DESCRIBE BookData;
```

- Cargar datos en la tabla:

```
LOAD DATA INPATH '/user/impala/input/BX-  
BooksCorrected.txt' OVERWRITE INTO TABLE BookData;
```

- Consulta 0: obtener el número de libros por cada año

```
SELECT YearOfPublication,  
COUNT(BookTitle) FROM BookData GROUP BY  
YearOfPublication;
```

- Discutid por grupos como se resolvería esta consulta en cada nodo (mappers) y cómo se combinarían los resultados (reducers).

- Una tabla *interna* se puede particionar, lo que significa que los registros se dividen en varios flujos en función de que tengan valores iguales en uno o varios atributos.
- Para crear una tabla particionada según un atributo específico:

```
CREATE TABLE IF NOT EXISTS BookDataPerYear (ISBN STRING,
BookTitle STRING, BookAuthor STRING, Publisher STRING)
PARTITIONED BY (YearOfPublication INT) ROW FORMAT
DELIMITED FIELDS TERMINATED BY '\;' STORED AS
TEXTFILE;
```

- Volcar datos a la tabla particionada desde la tabla sin partición:

```
INSERT OVERWRITE TABLE BookDataPerYear  
PARTITION (yearofpublication = 2003)  
SELECT ISBN, BookTitle, BookAuthor,  
Publisher FROM BookData WHERE  
yearofpublication = 2003;
```



- Inserción de registros por particiones:

```
INSERT OVERWRITE TABLE BookDataPerYear
PARTITION (yearofpublication = 2004)
SELECT ISBN, BookTitle, BookAuthor, Publisher FROM BookData WHERE
yearofpublication = 2004;
```

```
INSERT OVERWRITE TABLE BookDataPerYear
PARTITION (yearofpublication = 2005)
SELECT ISBN, BookTitle, BookAuthor, Publisher FROM BookData WHERE
yearofpublication = 2005;
```

```
INSERT OVERWRITE TABLE BookDataPerYear
PARTITION (yearofpublication = 2006)
SELECT ISBN, BookTitle, BookAuthor, Publisher FROM BookData WHERE
yearofpublication = 2006;
```

```
INSERT OVERWRITE TABLE BookDataPerYear
PARTITION (yearofpublication = 2007)
SELECT ISBN, BookTitle, BookAuthor, Publisher FROM BookData WHERE
yearofpublication = 2007;
```

- Los datos en las tablas BookData y BookDataPerYear para los años que están entre los años de publicación 2003 y 2007 deben ser los mismos. ¡Compruébalo!

```
SELECT YearOfPublication,  
       COUNT(BookTitle) FROM BookData WHERE  
       YearOfPublication BETWEEN 2003 AND 2007  
GROUP BY YearOfPublication;
```

```
SELECT YearOfPublication,  
       COUNT(BookTitle) FROM BookDataPerYear  
WHERE YearOfPublication BETWEEN 2003  
AND 2007 GROUP BY YearOfPublication;
```

```
SELECT COUNT(1) FROM (SELECT * FROM
BookDataPerYear WHERE YearOfPublication
BETWEEN 2003 AND 2007) bdpy LEFT SEMI
JOIN (SELECT * FROM BookData WHERE
YearOfPublication BETWEEN 2003 AND
2007) bd ON bdpy.ISBN = bd.ISBN;
```

Resultado: 20247 tuplas

- Consultar cuántos libros fueron publicados “en el año 0”:  

```
SELECT COUNT(1) FROM BookData WHERE YearOfPublication = 0;
```

- Limpiar la tabla BookData para quitar los libros del año 0:

```
INSERT OVERWRITE TABLE BookData
  SELECT BookData.* FROM BookData WHERE
  YearOfPublication > 0;
```

- ... y comprobamos:

```
SELECT COUNT(1) FROM BookData WHERE YearOfPublication = 0;
```

- Consultar el número de libros de cada autor, en la misma editorial y cada año:

```
SELECT Publisher, BookAuthor,  
       YearOfPublication, COUNT(BookTitle) FROM  
BookData GROUP BY Publisher, BookAuthor,  
YearOfPublication;
```