UNIT 2

DATA PREPROCESSING AND CLEANING

CONTENT

- Techniques
  - data cleaning
  - Transformation
  - integration
- Handling missing data and outliers in big data
- Data reduction and feature selection methods
- Scalable data preprocessingworkflows using Apache Pig and Apache Hive

DATA CLEANING IN BIG DATA ANALYTICS

The goal is to improve the reliability of the data so that the insights derived from it are accurate and actionable.

Common Data Issues in Big Data

- **Missing Values**: Missing data points in the dataset, which can lead to biased results if not handled properly.
- **Duplicate Entries**: Redundant records that can distort analysis by inflating the frequency of certain values.
- **Inconsistent Formats**: Data may be recorded in different formats across sources (e.g., date formats like MM/DD/YYYY vs. DD/MM/YYYY).
- **Outliers**: Extreme values that can skew statistical analyses and models.
- **Noise**: Irrelevant or extraneous data that can obscure meaningful patterns.

STEPS IN DATA CLEANING

01	02	03	04	05	06	07
Identifying Errors	Handling Missing Data	Removing Duplicates	Example with PySpark:	Standardizing Formats	Handling Outliers	Removing Noise

IDENTIFYING ERRORS

Data Profiling:

- Analyze the data to understand its structure, content, and quality. This includes checking for missing values, duplicates, and inconsistencies.
- **Apache Spark (with DataFrames and SQL):** Apache Spark, particularly with its DataFrame and SQL APIs, is a powerful tool for data profiling in Big Data environments. It can handle large datasets efficiently and provides various methods to explore the structure, content, and quality of the data.

Key Features:

- **Schema Exploration:** Inspect the schema to understand data types and column names.
- **Null Checks:** Use built-in functions to check for missing values in columns.
- **Duplicate Detection:** Easily identify and count duplicate records.
- **Summary Statistics:** Generate summary statistics (e.g., mean, median, standard deviation) for numeric columns to identify data distributions.

PYTHON CODE

```
from pyspark.sql import SparkSession

from pyspark.sql.functions import col, count, isnan, when

# Initialize Spark session

spark = SparkSession.builder.appName("DataProfilingExample").getOrCreate()

# Load data into a DataFrame

df = spark.read.csv("path/to/your/data.csv", header=True, inferSchema=True)

# Inspect schema

df.printSchema()
```

```
# Check for missing values

df.select([count(when(col(c).isNull() | isnan(c), c)).alias(c) for c in df.columns]).show()

# Identify duplicates

duplicates = df.groupBy(df.columns).count().filter("count > 1")

duplicates.show()

# Summary statistics

df.describe().show()
```

OTHER TOOLS

Talend Data Quality:

Talend Data Quality is a tool designed to profile data, allowing you to assess the quality and structure of your data through a user-friendly interface.

Key Features:

Data Profiling: Automatically detect data patterns, missing values, and duplicate records.

Data Quality Indicators: Get insights into the completeness, uniqueness, and validity of your data.

Visual Reports: Generate visual reports to understand data distributions and identify anomalies.

DataRobot Paxata:

DataRobot Paxata provides self-service data preparation capabilities with built-in data profiling features. It helps users quickly understand their data quality and structure.

Key Features:

Automated Data Profiling: Instantly analyze datasets to find missing values, outliers, and inconsistencies.

Interactive Exploration: Use a visual interface to drill down into data issues and understand their impact.

Smart Suggestions: Receive intelligent suggestions for data cleaning and transformation based on profiling results.

STATISTICAL ANALYSIS TOOLS

Python (Pandas & NumPy):

Python, with libraries like Pandas and NumPy, is widely used for statistical analysis.

Though more suitable for smaller datasets, these tools can be effective when working on subsets or when integrated into Big Data pipelines.

Key Features:

Descriptive Statistics: Use functions like mean(), median(), std(), and describe() to get an overview of the data distribution.

Outlier Detection: Identify outliers using methods like Z-scores or the Interquartile Range (IQR).

Visualization: Use Matplotlib or Seaborn for visualizing distributions and detecting unusual patterns.

PYTHON CODE

```
import pandas as pd

import numpy as np


# Load data into a DataFrame

df = pd.read_csv("path/to/your/data.csv")
```

```
# Descriptive statistics

print(df.describe())


# Detect outliers using Z-scores

df['z_score'] = (df['column_name'] -
df['column_name'].mean()) /
df['column_name'].std()

outliers = df[np.abs(df['z_score']) > 3]

print(outliers)
```

OTHER TOOLS

R (DataExplorer & summarytools):

R is another powerful tool for statistical analysis, especially in exploratory data analysis.

Packages like DataExplorer and summarytools provide comprehensive profiling and statistical summary features.

Key Features:

Data Summarization: Generate detailed summary statistics for all columns.

Correlation Analysis: Explore relationships between variables to identify unusual correlations.

Outlier Analysis: Use built-in functions to detect and visualize outliers.

OTHER TOOLS

IBM InfoSphere Information Analyzer:

IBM InfoSphere Information Analyzer provides advanced data profiling and analysis features for large-scale data environments, particularly in enterprise settings.

Key Features:

Data Quality Analysis: Automatically assess data quality and detect inconsistencies across datasets.

Outlier Detection: Identify and flag outliers and anomalies in large datasets.

Data Profiling Reports: Generate comprehensive reports that include statistical analysis and quality metrics.

HANDLING MISSING DATA

Removal:

If a significant portion of data is missing in a row or column, you might choose to remove those records.

Imputation:

Replace missing values with a substitute value, such as the mean, median, or mode of the column, or use more advanced methods like predictive modeling.

Flagging:

Mark missing values with a specific flag so they can be accounted for in the analysis.

CODE

```
# Dropping rows with missing values
df_cleaned = df.dropna()

# Imputing missing values with the mean
from pyspark.sql.functions import mean
mean_value = df.select(mean(df['column_name'])).collect()[0][0]
df_cleaned = df.na.fill({'column_name': mean_value})
```

REMOVING DUPLICATES

- **Detection**: Identify duplicate records that appear multiple times in the dataset.
- **Removal**: Remove these duplicate records to ensure each entry is unique.

```
# Removing duplicate rows based on all columns
df_cleaned = df.dropDuplicates()

# Removing duplicates based on specific columns
df_cleaned = df.dropDuplicates(['column1', 'column2'])
```

STANDARDIZING FORMATS

Normalization:

Convert data into a consistent format, such as standardizing date formats or text case.

Mapping:

Use mapping tables or functions to convert categorical data into a standardized set of values.

```
from pyspark.sql.functions import to_date

# Standardizing date format
df_cleaned = df.withColumn('date_column', to_date(df['date_column'], 'MM/dd/yyyy'))
```

HANDLING OUTLIERS

- **Filtering**: Remove outliers if they are determined to be erroneous or irrelevant.
- **Capping**: Replace outlier values with a defined maximum or minimum threshold.
- **Transformation**: Apply mathematical transformations (e.g., logarithmic transformation) to reduce the impact of outliers.

```
# Filtering out outliers based on a threshold
df_cleaned = df.filter((df['column_name'] > lower_bound) & (df['column_name'] < upper_bound))
```

REMOVING NOISE

Filtering:

Exclude irrelevant data or columns that do not contribute to the analysis.

Dimensionality Reduction:

Techniques like Principal Component Analysis (PCA) can be used to reduce the number of variables while retaining the most important information.

APACHE PIG

A SCRIPTING TOOL

high-level platform for creating programs that run on Apache Hadoop.

Pig provides a simple interface for processing and analyzing large datasets in a distributed environment.

Pig is built on top of Hadoop's MapReduce framework, but it abstracts away the complexities, allowing users to focus on data processing without having to write complex MapReduce code.

It is highly suited for handling large-scale data processing tasks such as ETL (Extract, Transform, Load), data cleansing, and analysis.

WHAT IS PIG?

Developed by Yahoo Research and a top-level Apache project

Immediately makes data on a cluster available to non-Java programmers via Pig Latin – a dataflow language

Interprets Pig Latin and generates MapReduce jobs that run on the cluster

Enables easy data summarization, ad-hoc reporting and querying, and analysis of large volumes of data

Pig interpreter runs on a client machine – no administrative overhead required

KEY COMPONENTS OF PIG

Pig Latin Language:

- Pig Latin is the scripting language used in Apache Pig.
- It is designed to express data flows, much like SQL in relational databases, but with a more flexible and procedural syntax.
- Pig Latin scripts consist of statements that load, transform, and store data.
- Each operation creates a new data relation (like a table in SQL).

Pig Runtime Environment:

- This is where the Pig scripts are executed.
- Pig scripts are converted into MapReduce jobs and executed on the Hadoop cluster.

KEY COMPONENTS OF PIG

Execution Modes:

- **Local Mode**:
  - In local mode, Pig runs on a single machine and processes data stored in the local file system.
  - This mode is useful for small datasets or for testing Pig scripts.
- **MapReduce Mode**:
  - In MapReduce mode, Pig interacts with the Hadoop cluster, and data is processed in parallel across the nodes of the cluster.
  - This is the most common mode used for large-scale data processing.

Pig Grunt Shell:

- The Grunt shell is an interactive shell for executing Pig commands and running Pig scripts step-by-step. This is useful for debugging and development purposes.

FEATURES OF APACHE PIG

Ease of Use

- Pig Latin is simpler than Java-based MapReduce code, making it accessible to data engineers and analysts without requiring advanced programming skills.
- It allows users to focus on the logic of data transformation rather than the underlying mechanics of MapReduce.

Flexibility

- Pig supports complex, nested data types such as tuples, bags (collections of tuples), and maps. This allows Pig to handle structured, semi-structured, and unstructured data.
- Pig can be extended with User Defined Functions (UDFs) written in Java, Python, or other languages, allowing for custom operations beyond the standard library.

Optimization

- Pig optimizes execution by automatically converting Pig Latin scripts into a directed acyclic graph (DAG) of MapReduce jobs, which are then executed in a way that minimizes disk I/O and improves performance.
- The optimizer combines related operations to reduce the number of MapReduce jobs, making the process more efficient.

FEATURES OF APACHE PIG

Handles Various Data Sources:

- Pig can read data from HDFS, HBase, and other storage systems.
- It can handle a variety of data formats, including text files, CSV, JSON, and binary files.

Data Flow Model:

- Unlike SQL, Pig Latin is designed for data flows and transformations, making it more procedural.
- This procedural nature allows for complex sequences of operations, like multiple joins and filtering steps, to be written more intuitively than SQL.

Data Transformation:

- Pig allows users to perform a range of data transformations such as filtering, grouping, joining, and sorting.
- These operations are similar to SQL but with greater flexibility in handling complex data formats.

COMMON PIG LATIN OPERATORS

LOAD Reads data from the source (e.g., HDFS).	• data = LOAD 'data.txt' USING PigStorage(',') AS (id:int, name:chararray, age:int);
FILTER Removes unwanted data based on a condition.	• filtered_data = FILTER data BY age > 30;
FOREACH…GENERATE Projects specific columns or computes new data from the existing data.	• selected_data = FOREACH data GENERATE id, name;
GROUP Groups the data by a particular field or column.	• grouped_data = GROUP data BY city;
JOIN Combines two or more datasets based on a common field.	• joined_data = JOIN data1 BY id, data2 BY id;
ORDER BY Sorts data by a field.	• sorted_data = ORDER data BY age DESC;
STORE Saves the processed data to a specified location, typically in HDFS.	• STORE filtered_data INTO 'output' USING PigStorage(',');

USE CASES OF APACHE PIG

ETL (EXTRACT, TRANSFORM, LOAD):

- Apache Pig is widely used for ETL tasks such as extracting raw data from different sources, transforming it by filtering, aggregating, or joining datasets, and then loading it into the desired destination (e.g., HDFS, HBase).

```
-- Load raw data from a CSV file
raw_data = LOAD 'data.csv' USING PigStorage(',') AS (id:int,
name:chararray, age:int, city:chararray);
-- Filter records to select people above the age of 25
filtered_data = FILTER raw_data BY age > 25;
-- Group data by city
grouped_data = GROUP filtered_data BY city;
-- Store the transformed data
STORE grouped_data INTO 'output_data' USING PigStorage(',');
```

DATA CLEANSING:

- Pig is used to clean messy or incomplete data by handling null values, removing duplicates, or standardizing formats.

```
-- Load dirty data
dirty_data = LOAD 'dirty_data.txt' USING PigStorage(',') AS
(id:int, name:chararray, age:int);


-- Remove records with missing age
clean_data = FILTER dirty_data BY age IS NOT NULL;
```

DATA ANALYSIS

- Pig is highly effective in performing large-scale data analysis tasks, such as analyzing server logs, sensor data, or social media feeds.

```
-- Load server logs
logs = LOAD 'logs.txt' USING PigStorage(' ') AS (timestamp:chararray,
user_id:chararray, action:chararray);


-- Count the number of actions performed by each user
user_actions = GROUP logs BY user_id;
action_count = FOREACH user_actions GENERATE group AS user_id, COUNT(logs) AS
num_actions;


-- Store the result
STORE action_count INTO 'user_action_count' USING PigStorage(',');
```

HANDLING SEMI-STRUCTURED DATA:

Pig can work with semi-structured data such as JSON or XML.

It’s widely used to process log files and other types of semi-structured or unstructured data, making it highly flexible.

PIG TERMS

An Atom is a simple data value - stored as a string but can be used as either a string or a number

A Tuple is a data record consisting of a sequence of "fields"

- Each field is a piece of data of any type (atom, tuple or bag)

A Bag is a set of tuples (also referred to as a ‘Relation’)

- The concept of a “kind of a” table

A Map is a map from keys that are string literals to values that can be any data type

- The concept of a hash map

SIMPLE DATA TYPES

Type	Description
int	4-byte integer
long	8-byte integer
float	4-byte (single precision) floating point
double	8-byte (double precision) floating point
bytearray	Array of bytes; blob
chararray	String (“hello world”)
boolean	True/False (case insensitive)
datetime	A date and time
biginteger	Java BigInteger
bigdecimal	Java BigDecimal

COMPLEX DATA TYPES

Type	Description
Tuple	Ordered set of fields (a “row / record”)
Bag	Collection of tuples (a “resultset / table”)
Map	A set of key-value pairs Keys must be of type chararray

PIG DATA FORMATS

BinStorage	PigStorage	TextLoader	PigDump	YourOwnFormat!
• Loads and stores data in machine-readable (binary) format	• Loads and stores data as structured, field delimited text files	• Loads unstructured data in UTF-8 format	• Stores data in UTF-8 format	• via UDFs

DIFFERENCE BETWEEN PIG AND HIVE

Feature	Apache Pig	Apache Hive
Language	Pig Latin (Procedural Language)	HiveQL (Declarative Language, similar to SQL)
Purpose	Data transformation and processing (especially for ETL tasks).	Data querying, reporting, and analysis.
Data Model	Works well with semi-structured data (e.g., JSON, log files).	Best suited for structured data, similar to relational databases.
Type of Processing	Procedural, where each step defines how the data is transformed.	Declarative, users specify what to do with data (similar to SQL).
Ease of Learning	Requires learning Pig Latin, which is more procedural and script-like.	Easier for users familiar with SQL, as it uses SQL-like queries (HiveQL).
Use Case	Complex ETL (Extract, Transform, Load), data cleansing, and transformations.	Querying, reporting, and data analysis (OLAP-like operations).
Execution	Converts Pig Latin scripts into MapReduce jobs.	Converts HiveQL queries into MapReduce, Tez, or Spark jobs.
Optimization	Built-in optimizations but more manual effort may be required.	Query optimization engine (Cost-based optimization in recent versions).

DIFFERENCE BETWEEN PIG AND HIVE

Feature	Apache Pig	Apache Hive
Support for UDFs	Supports User Defined Functions (UDFs) written in Java, Python, etc.	Supports UDFs written in Java, Python, and more, using Hive UDFs.
Schema on Read/Write	Does not require a fixed schema before loading data.	Uses schema-on-read; schema is defined when the table is created.
Target Users	Developers and engineers who need custom data transformation flows.	Analysts and data scientists who are familiar with SQL-like queries.
Execution Modes	Local mode (on a single machine) and MapReduce mode (on Hadoop).	Hive supports execution on Hadoop using MapReduce, Tez, and Spark.
Performance	Efficient for complex data transformations but can be slower than Hive for simple queries.	Faster for simple SQL queries, especially when using Tez or Spark as the execution engine.

DIFFERENCE BETWEEN PIG AND HIVE

Feature	Apache Pig	Apache Hive
Use of Joins	More flexible but less optimized for complex joins compared to Hive.	Better optimized for performing complex joins and aggregations.
Community & Ecosystem	Originally developed by Yahoo! for data pipelines. Pig has a niche community.	Larger community and broader ecosystem, as it is widely used by data analysts.
Execution Engine	Pig runs on the MapReduce engine (with support for Spark in recent versions).	Hive can run on MapReduce, Tez, or Spark, making it more flexible for different workloads.
Error Handling	Debugging can be more challenging due to its procedural nature.	Easier to debug since it is more SQL-like and declarative.
Example Use Case	Data transformation in ETL pipelines, log analysis, sessionization of data.	Querying large datasets for reporting, aggregating, and summarizing data.

APACHE HIVE

BASED ON SLIDES BY ADAM SHOOK

WHAT IS HIVE?

- Developed by Facebook and a top-level Apache project
- A data warehousing infrastructure based on Hadoop
- Immediately makes data on a cluster available to non-Java programmers via SQL like queries
- Built on HiveQL (HQL), a SQL-like query language
- Interprets HiveQL and generates MapReduce jobs that run on the cluster
- Enables easy data summarization, ad-hoc reporting and querying, and analysis of large volumes of data

WHAT HIVE IS NOT

- Hive, like Hadoop, is designed for batch processing of large datasets
- Not an OLTP or real-time system
- Latency and throughput are both high compared to a traditional RDBMS
  - Even when dealing with relatively small data (<100 MB)

DATA HIERARCHY

- Hive is organised hierarchically into:
  - Databases: namespaces that separate tables and other objects
  - Tables: homogeneous units of data with the same schema
    - Analogous to tables in an RDBMS
  - Partitions: determine how the data is stored
    - Allow efficient access to subsets of the data
  - Buckets/clusters
    - For sub-sampling within a partition
    - Join optimization

HIVEQL

- HiveQL / HQL provides the basic SQL-like operations:
  - Select columns using SELECT
  - Filter rows using WHERE
  - JOIN between tables
  - Evaluate aggregates using GROUP BY
  - Store query results into another table
  - Download results to a local directory (i.e., export from HDFS)
  - Manage tables and queries with CREATE, DROP, and ALTER

PRIMITIVE DATA TYPES

Type	Comments
TINYINT, SMALLINT, INT, BIGINT	1, 2, 4 and 8-byte integers
BOOLEAN	TRUE/FALSE
FLOAT, DOUBLE	Single and double precision real numbers
STRING	Character string
TIMESTAMP	Unix-epoch offset *or* datetime string
DECIMAL	Arbitrary-precision decimal
BINARY	Opaque; ignore these bytes

COMPLEX DATA TYPES

Type	Comments
STRUCT	A collection of elements If S is of type STRUCT {a INT, b INT}: S.a returns element a
MAP	Key-value tuple If M is a map from 'group' to GID: M['group'] returns value of GID
ARRAY	Indexed list If A is an array of elements ['a','b','c']: A[0] returns 'a'

HIVEQL LIMITATIONS

- HQL only supports equi-joins, outer joins, left semi-joins
- Because it is only a shell for Map-Reduce, complex queries can be hard to optimise
- Missing large parts of full SQL specification:
  - HAVING clause in SELECT
  - Correlated sub-queries
  - Sub-queries outside FROM clauses
  - Updatable or materialized views
  - Stored procedures

HIVE METASTORE

- Stores Hive metadata
- Default metastore database uses Apache Derby
- Various configurations:
  - Embedded (in-process metastore, in-process database)
    - Mainly for unit tests
  - Local (in-process metastore, out-of-process database)
    - Each Hive client connects to the metastore directly
  - Remote (out-of-process metastore, out-of-process database)
    - Each Hive client connects to a metastore server, which connects to the metadata database itself

HIVE WAREHOUSE

- Hive tables are stored in the Hive “warehouse”
  - Default HDFS location: /user/hive/warehouse
- Tables are stored as sub-directories in the warehouse directory
- Partitions are subdirectories of tables
- External tables are supported in Hive
- The actual data is stored in flat files

HIVE SCHEMAS

- Hive is schema-on-read
  - Schema is only enforced when the data is read (at query time)
  - Allows greater flexibility: same data can be read using multiple schemas
- Contrast with an RDBMS, which is schema-on-write
  - Schema is enforced when the data is loaded
  - Speeds up queries at the expense of load times

CREATE TABLE SYNTAX

```
CREATE TABLE table_name
    (col1 data_type,
     col2 data_type,
     col3 data_type,
     col4 datatype )
   ROW FORMAT DELIMITED
   FIELDS TERMINATED BY ','
   STORED AS format_type;
```

SIMPLE TABLE

```
CREATE TABLE page_view
    (viewTime INT,
     userid BIGINT,
     page_url STRING,
     referrer_url STRING,
     ip STRING COMMENT 'IP Address of the User' )
   ROW FORMAT DELIMITED
   FIELDS TERMINATED BY '\t'
   STORED AS TEXTFILE;
```

MORE COMPLEX TABLE

```
CREATE TABLE employees  (
     (name STRING,
     salary FLOAT,
     subordinates ARRAY<STRING>,
     deductions MAP<STRING, FLOAT>,
     address STRUCT<street:STRING,
                    city:STRING,
                    state:STRING,
                    zip:INT>)
   ROW FORMAT DELIMITED
   FIELDS TERMINATED BY '\t'
   STORED AS TEXTFILE;
```

EXTERNAL TABLE

```
CREATE EXTERNAL TABLE page_view_stg
    (viewTime INT,
     userid BIGINT,
     page_url STRING,
     referrer_url STRING,
     ip STRING COMMENT 'IP Address of the User')
   ROW FORMAT DELIMITED
   FIELDS TERMINATED BY '\t'
   STORED AS TEXTFILE
   LOCATION '/user/staging/page_view';
```

MORE ABOUT TABLES

- CREATE TABLE
  - LOAD: file moved into Hive's data warehouse directory
  - DROP: both metadata and data deleted
- CREATE EXTERNAL TABLE
  - LOAD: no files moved
  - DROP: only metadata deleted
  - Use this when sharing with other Hadoop applications, or when you want to use multiple schemas on the same data

PARTITIONING

- Can make some queries faster
- Divide data based on partition column
- Use PARTITION BY clause when creating table
- Use PARTITION clause when loading data
- SHOW PARTITIONS will show a table's partitions

BUCKETING

- Can speed up queries that involve sampling the data
  - Sampling works without bucketing, but Hive has to scan the entire dataset
- Use CLUSTERED BY when creating table
  - For sorted buckets, add SORTED BY
- To query a sample of your data, use TABLESAMPLE

BR S

Command	Comments
`SHOW TABLES;`	Show all the tables in the database
`SHOW TABLES 'page.*';`	Show tables matching the specification (uses regex syntax)
`SHOW PARTITIONS page_view;`	Show the partitions of the page_view table
`DESCRIBE page_view;`	List columns of the table
`DESCRIBE EXTENDED page_view;`	More information on columns (useful only for debugging)
`DESCRIBE page_view PARTITION (ds='2008-10-31');`	List information about a partition

LOADING DATA

- Use LOAD DATA to load data from a file or directory
  - Will read from HDFS unless LOCAL keyword is specified
  - Will append data unless OVERWRITE specified
  - PARTITION required if destination table is partitioned

```
LOAD DATA LOCAL INPATH '/tmp/pv_2008-06-8_us.txt'
  OVERWRITE INTO TABLE page_view
  PARTITION (date='2008-06-08', country='US')
```

INSERTING DATA

- Use INSERT to load data from a Hive query
  - Will append data unless OVERWRITE specified
    - PARTITION required if destination table is partitioned

```
FROM page_view_stg pvs
 INSERT OVERWRITE TABLE page_view
 PARTITION (dt='2008-06-08', country='US')
  SELECT pvs.viewTime, pvs.userid, pvs.page_url,
         pvs.referrer_url
 WHERE pvs.country = 'US';
```

LOADING AND INSERTING DATA: SUMMARY

Use this	For this purpose
`LOAD`	Load data from a file or directory
`INSERT`	Load data from a query • One partition at a time • Use multiple INSERTs to insert into multiple partitions in the one query
`CREATE TABLE AS (CTAS)`	Insert data while creating a table
Add/modify external file	Load new data into external table

SAMPLE SELECT CLAUSES

- Select from a single table

```
SELECT *
        FROM sales
        WHERE amount > 10 AND
                                region = "US";
```

- Select from a partitioned table

```
SELECT page_views.*
FROM page_views
WHERE page_views.date >= '2008-03-01' AND
      page_views.date <= '2008-03-31'
```

RELATIONAL OPERATORS

- ALL and DISTINCT
  - Specify whether duplicate rows should be returned
  - ALL is the default (all matching rows are returned)
  - DISTINCT removes duplicate rows from the result set
- WHERE
  - Filters by expression
  - Does not support IN, EXISTS or sub-queries in the WHERE clause
- LIMIT
  - Indicates the number of rows to be returned

RELATIONAL OPERATORS

- GROUP BY
  - Group data by column values
  - Select statement can only include columns included in the GROUP BY clause
- ORDER BY / SORT BY
  - ORDER BY performs total ordering
    - Slow, poor performance
  - SORT BY performs partial ordering
    - Sorts output from each reducer

ADVANCED HIVE OPERATIONS

- JOIN
  - If only one column in each table is used in the join, then only one MapReduce job will run
    - This results in 1 MapReduce job:

```
SELECT * FROM a JOIN b ON a.key = b.key JOIN c ON b.key = c.key
```

    - This results in 2 MapReduce jobs:

```
SELECT * FROM a JOIN b ON a.key = b.key JOIN c ON b.key2 = c.key
```

  - If multiple tables are joined, put the biggest table last and the reducer will stream the last table, buffer the others
  - Use left semi-joins to take the place of IN/EXISTS

```
SELECT a.key, a.val FROM a LEFT SEMI JOIN b on a.key = b.key;
```

ADVANCED HIVE OPERATIONS

- JOIN
  - Do not specify join conditions in the WHERE clause
    - Hive does not know how to optimise such queries
    - Will compute a full Cartesian product before filtering it
- Join Example

```
SELECT
  a.ymd, a.price_close, b.price_close
FROM stocks a
JOIN stocks b ON a.ymd = b.ymd
WHERE a.symbol = 'AAPL' AND
      b.symbol = 'IBM' AND
      a.ymd > '2010-01-01';
```

HIVE STINGER

- MPP-style execution of Hive queries
- Available since Hive 0.13
- No MapReduce
- We will talk about this more when we get to SQL on Hadoop

REFERENCES

- http://hive.apache.org