**U.V. PATEL COLLEGE OF ENGINEERING**

**Department of Computer Engineering and Information Technology**

**Subject: Big Data Analytics (2CE709/2IT709)**

**LAB-1**

**Task-1:**

Consider the usage examples of Big Data for a car company. Assume that company manufactures five models of cars, and each model is available in five colours and five shades. The company collects inputs from customers and sales centres, and inputs of component malfunctions from service centres for different models. The company also uses social media inputs.

Explain 3Vs characteristics in this company's data.

**Task-2:**

Study the Drivers of Big Data and Summarize your Understanding.

**Task-3:**

Study the Data Formats given below and summarize your understanding

1. CSV / TSV
2. JSON
3. Parquet
4. Avro
5. Key-Value Pair

**CSV**

CSV files (comma separated values) are commonly used to exchange tabular data between systems using plain text. CSV is a row-based file format, which means that every line of the file is the row in the table. Basically, CSV contains a header row that provides column names for the data, otherwise, files are considered partially structured. CSV files cannot initially present hierarchical or relational data. Data connections are typically organized by using multiple CSV files. Foreign keys are stored in columns of one or more files, but the links between these files are not expressed by the format itself. Also, CSV format is not fully standardized, files can use delimiters other than commas, such as tabs or spaces.

One of the other properties of CSV files is that they are only splittable when it is a raw, uncompressed file or when splittable compression format is used such as [bzip2](https://en.wikipedia.org/wiki/Bzip2) or [lzo](https://ru.wikipedia.org/wiki/LZO" \t "_blank) (note: lzo needs to be indexed to be splittable!).

**Advantages**:

* CSV is human-readable and easy to edit manually;
* CSV provides a straightforward information schema;
* CSV is processed by almost all existing applications;
* CSV is simple to implement and parse;
* CSV is compact. For XML you start tag and end tag for each column in each row. In CSV you write the column headers only once.

**Disadvantages**:

* CSV allows working with flat data. Complex data structures need to be handled aside from format;
* No support for column types. There is no distinction between text and numeric columns;
* No standard way to represent binary data;
* Problems with importing CSV (no distinction between NULL and quotes);
* Poor support of special characters;
* Lack of universal standard.

Despite the limitations, CSV files are a popular choice for data sharing, as they are supported by a wide range of business applications, consumer and scientific applications. Similarly, most batch and streaming data processing modules (for example, Spark and Hadoop) initially support serialization and deserialization of CSV files and offer ways to add a schema when reading.

# JSON

JSON data (JavaScript object notation) is represented as key-value pairs in a partially structured format. JSON is often compared to XML because it can store data in a hierarchical format. Child data is presented by parent data. Both formats are self-describing and readable by the user, but JSON documents are usually much smaller. Therefore, they are more often used in network communication, especially with the advent of REST-based web services.

Since a lot of data transferring is already using JSON format, most web languages initially support JSON or use external libraries to serialize and deserialize JSON data. Thanks to this support, JSON is used in logical formats by presenting data structures, interchange formats for hot data, and cold data stores.

Many batches and stream data processing modules natively support JSON serialization and deserialization. Although the data contained in JSON documents can ultimately be stored in more performance-optimized formats, such as Parquet or Avro, they serve as raw data, which is very important for reprocessing data (if necessary).

JSON files have several **advantages**:

* JSON supports hierarchical structures, simplifying the storage of related data in one document and the presentation of complex relations;
* Most languages provide simplified JSON serialization libraries or built-in support for JSON serialization/deserialization;
* JSON supports lists of objects, helping to avoid erratic transformations of lists into a relational data model;
* JSON is a widely used file format for NoSQL databases such as MongoDB, Couchbase and Azure Cosmos DB;
* Built-in support in most nowadays tools.

# Parquet

Launched in 2013, Parquet was developed by Cloudera and Twitter to serve as a column-based storage format, optimized for work with multi-column datasets. Because data is stored by columns, it can be highly compressed (compression algorithms perform better on data with low information entropy which is usually contained in columns) and splittable. The developers of the format claim that this storage format is ideal for Big Data problems.

Unlike CSV and JSON, Parquet files are **binary** files that contain **metadata** about their content. So, without reading/parsing the content of the file(s), Spark can just rely on the metadata to determine column names, compression/encodings, data types and even some basic statistics. The column metadata for a Parquet file is stored at the end of the file, which allows for fast, one-pass writing.

Parquet is optimized for the [Write Once Read Many (WORM)](https://en.wikipedia.org/wiki/Write_once_read_many) paradigm. It’s slow to write, but incredibly fast to read, especially when you’re only accessing a subset of the total columns. Parquet is a **good choice for read-heavy workloads**. For use cases requiring operating on entire rows of data, a format like CSV or AVRO should be used.

The **advantages** of data storage in Parquet:

* Parquet is a columnar format. Only required columns would be fetched/read, it reduces the disk I/O. This concept is called projection pushdown;
* Schema travels with the data so data is self-describing;
* Despite the fact that it is created for HDFS, data can be stored in other file systems, such as GlusterFs or on top of NFS;
* Parquet are just files, which means that it is easy to work with them, move, back up and replicate;
* Native support in Spark out of the box provides the ability to simply take and save the file to your storage;
* Parquet provides very good compression up to 75% when used even with the compression formats like [snappy](https://en.wikipedia.org/wiki/Snappy_(compression));
* As practice shows, this format is the fastest for reading workflows compared to other file formats;
* Parquet is well suited for data warehouse kind of solutions where aggregations are required on certain column over a huge set of data;
* Parquet can be read and write using Avro API and Avro Schema(which gives the idea to store all raw data in Avro format but all processed data in Parquet);
* It also provides predicate pushdown, thus reducing further disk I/O cost.

**Disadvantages**:

* The column-based design makes you think about the schema and data types;
* Parquet does not always have native support in other tools other than Spark;
* Does not support data modification and schematic evolution. Of course, Spark knows how to merge the scheme, if you change it over time (you need to specify a special option when reading). But to change something in an already existing file, you can do nothing other than overwriting, except that you can add a new column.

# Avro

Apache Avro was released by the Hadoop working group in 2009. It is a **row-based format** that is highly splittable. It also described as a data serialization system similar to Java Serialization. The schema is stored in JSON format while the data is stored in binary format, minimizing file size and maximizing efficiency. Avro has robust support for **schema evolution** by managing added fields, missing fields, and fields that have changed. This allows old software to read the new data and new software to read the old data — a critical feature if your data has the potential to change.

With Avro’s capacity to manage schema evolution, it’s possible to update components independently, at different times, with low risk of incompatibility. This saves applications from having to write if-else statements to process different schema versions and saves the developer from having to look at old code to understand old schemas. Because all versions of the schema are stored in a human-readable JSON header, it’s easy to understand all the fields that you have available.

Avro can support many different programming languages. Because the schema is stored in JSON while the data is in binary, Avro is a relatively compact option for both persistent data storage and wire transfer. Avro is typically the format of choice for write-heavy workloads given its easy to append new rows.

**Advantages**:

* Avro is language-neutral data serialization
* Avro stores the schema in the header of the file so data is self-describing;
* Avro formatted files are splittable and compressible and hence it’s a good candidate for data storage in Hadoop ecosystem;
* The schema used to read an Avro file need not be the same as schema which was used to write the files. This makes it possible to add new fields independently.
* Just as with Sequence Files, Avro files also contains Sync markers to separate the blocks. This makes it highly splittable;
* These blocks can be compressed using compression formats such as snappy.