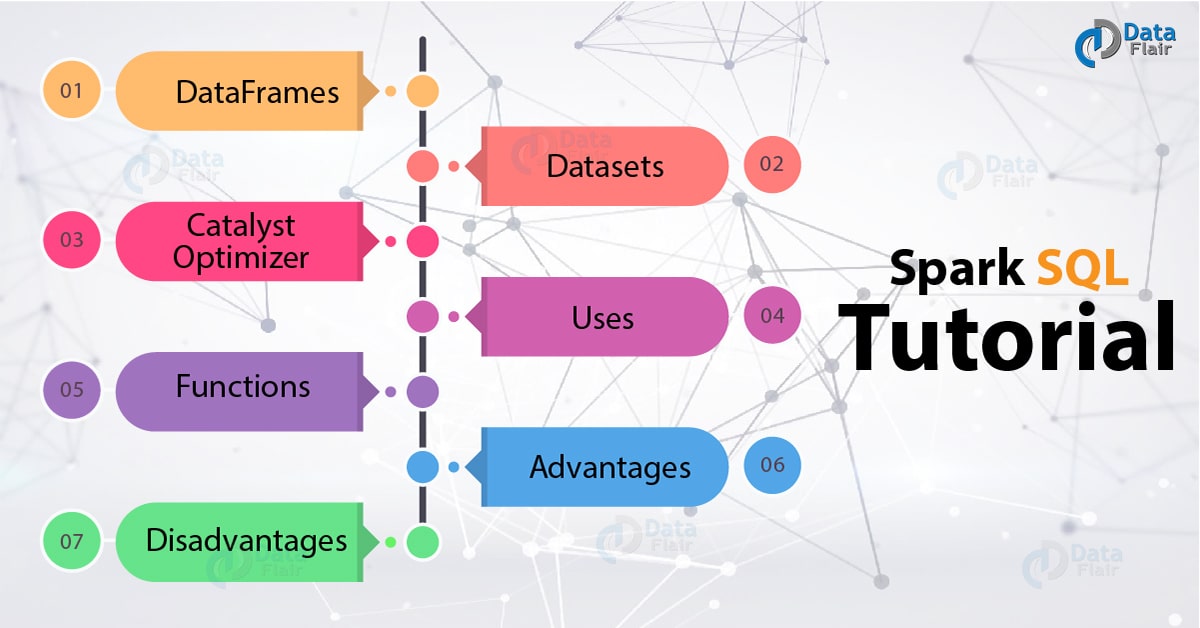
[**Apache Spark**](http://data-flair.training/blogs/apache-spark-tutorial-quickstart-introduction/) **SQL** is a Spark module to simplify working with structured data using DataFrame and DataSet abstractions in Python, Java, and **[Scala](http://data-flair.training/blogs/why-you-should-learn-scala-introductory-tutorial/)**. These abstractions are the distributed collection of data organized into named columns. It provides a good optimization technique. Using Spark SQL we can query data, both from inside a Spark program and from external tools that connect through standard database connectors (JDBC/ODBC) to Spark SQL.



**i. What is Spark SQL?**

**Apache Spark SQL** is a module for structured data processing in Spark. Using the interface provided by Spark SQL we get more information about the structure of the data and the computation performed. With this extra information, one can achieve extra optimization in Apache Spark. We can interact with Spark SQL in various ways like [**DataFrame**](http://data-flair.training/blogs/apache-spark-sql-dataframe-tutorial/) and the [**Dataset API**](http://data-flair.training/blogs/apache-spark-dataset-tutorial/). The Same execution engine is used while computing a result, irrespective of which API/language we use to express the computation. Thus, the user can easily switch back and forth between different APIs, it provides the most natural way to express a given transformation.  
In Apache Spark SQL we can use structured and semi-structured data in three ways:

* To simplify working with structured data it provides DataFrame abstraction in *Python, Java, and* [***Scala***](http://data-flair.training/blogs/why-you-should-learn-scala-introductory-tutorial/). DataFrame is a distributed collection of data organized into named columns. It provides a good optimization technique.
* The data can be read and written in a variety of structured formats. For example, JSON, [**Hive**](http://data-flair.training/blogs/apache-hive-tutorial-introductory-guide/)Tables, and Parquet.
* Using SQL we can query data, both from inside a Spark program and from external tools. The external tool connects through standard database connectors (JDBC/ODBC) to Spark SQL.
* The best way to use Spark SQL is inside a Spark application. This empowers us to load data and query it with SQL. At the same time, we can also combine it with “regular” program code in Python, Java or Scala.

When SQL run from the other programming language the result will be a **Dataset/DataFrame**. The interaction with SQL interface is made using the command line or over JDBC/ODBC.

**ii. Spark SQL DataFrames**

There were some [**limitations with RDDs.**](http://data-flair.training/blogs/apache-spark-rdd-limitations/) When working with structured data, there was no inbuilt optimization engine. On the basis of attributes, the developer optimized each [**RDD**](http://data-flair.training/blogs/apache-spark-rdd-tutorial/)**.** Also, there was no provision to handle structured data. The DataFrame in Spark SQL overcomes these limitations of RDD. **Spark DataFrame** is Spark 1.3 release. It is a distributed collection of data ordered into named columns. Concept wise it is equal to the table in a relational database or a data frame in [**R**](http://data-flair.training/blogs/r-programming-tutorial/)/Python. We can create DataFrame using:

* Structured data files
* Tables in Hive
* External databases
* Using existing RDD

### iii. Spark SQL Datasets

**Spark Dataset** is an interface added in version Spark 1.6. it is a distributed collection of data. Dataset provides the [**benefits of RDDs**](http://data-flair.training/blogs/apache-spark-rdd-features/) along with the benefits of Apache Spark SQL’s optimized execution engine. Here an encoder is a concept that does conversion between JVM objects and tabular representation.  
A Dataset can be made using JVM objects and after that, it can be manipulated using functional transformations (map, filter etc.). The Dataset API is accessible in Scala and Java. Dataset API is not supported by Python. But because of the dynamic nature of Python, many benefits of Dataset API are available. The same is the case with **R**. Using a Dataset of rows we represent DataFrame in Scala and Java.

**iv. Spark Catalyst Optimizer**

The *optimizer* used by Spark SQL is **Catalyst optimizer**. It optimizes all the queries written in Spark SQL and DataFrame DSL. The optimizer helps us to run queries much faster than their counter RDD part. This increases the performance of the system.  
**Spark Catalyst** is a library built as a rule-based system. And each rule focusses on the specific optimization. For example, *ConstantFolding* focus on eliminating *constant expression* from the query.

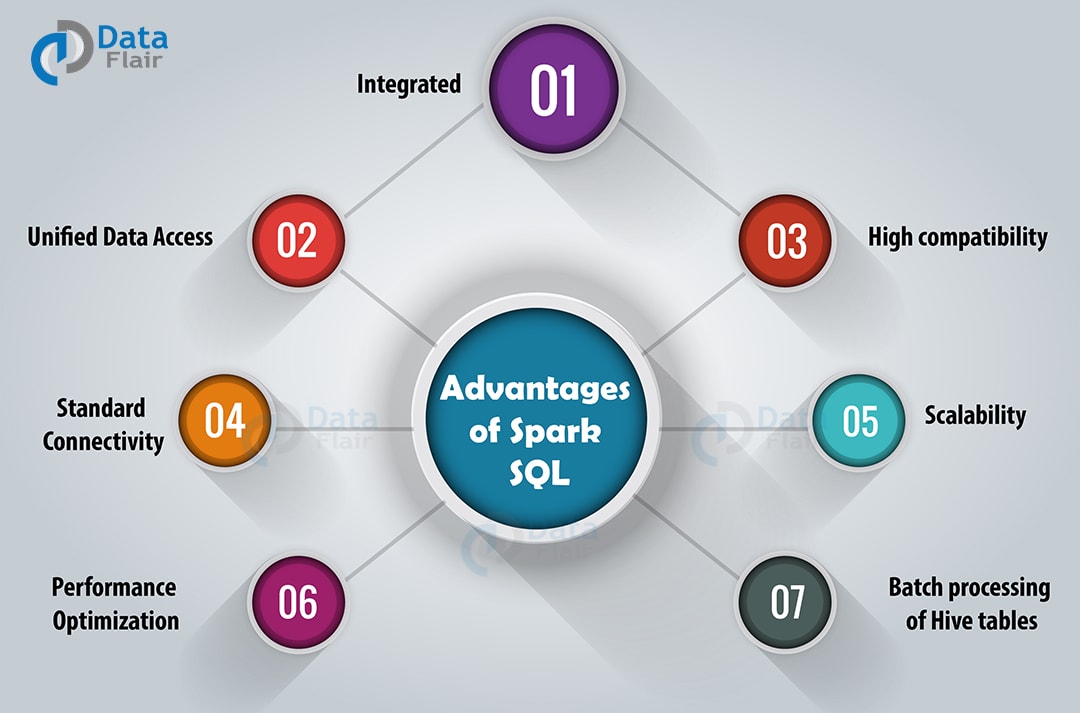
**v. Uses of Apache Spark SQL**

* It executes SQL queries.
* We can read data from existing [**Hive installation**](http://data-flair.training/blogs/apache-hive-installation-tutorial/) using SparkSQL.
* When we run SQL within another programming language we will get the result as Dataset/DataFrame.

**vi. Functions defined by Spark SQL**

**a. Built-In function**  
It offers a built-in function to process the column value. We can access the inbuilt function by importing the following command: ***Import org.apache.spark.sql.functions***  
**b. User Defined Functions(UDFs)**  
UDF allows you to create the user define functions based on the user-defined functions in Scala. Refer this guide to [learn the features of Scala.](http://data-flair.training/blogs/scala-features-comprehensive-guide/)  
**c. Aggregate functions**  
These operate on a group of rows and calculate a single return value per group.  
**d. Windowed Aggregates(Windows)**  
These operate on a group of rows and calculate a single return value for each row in a group.

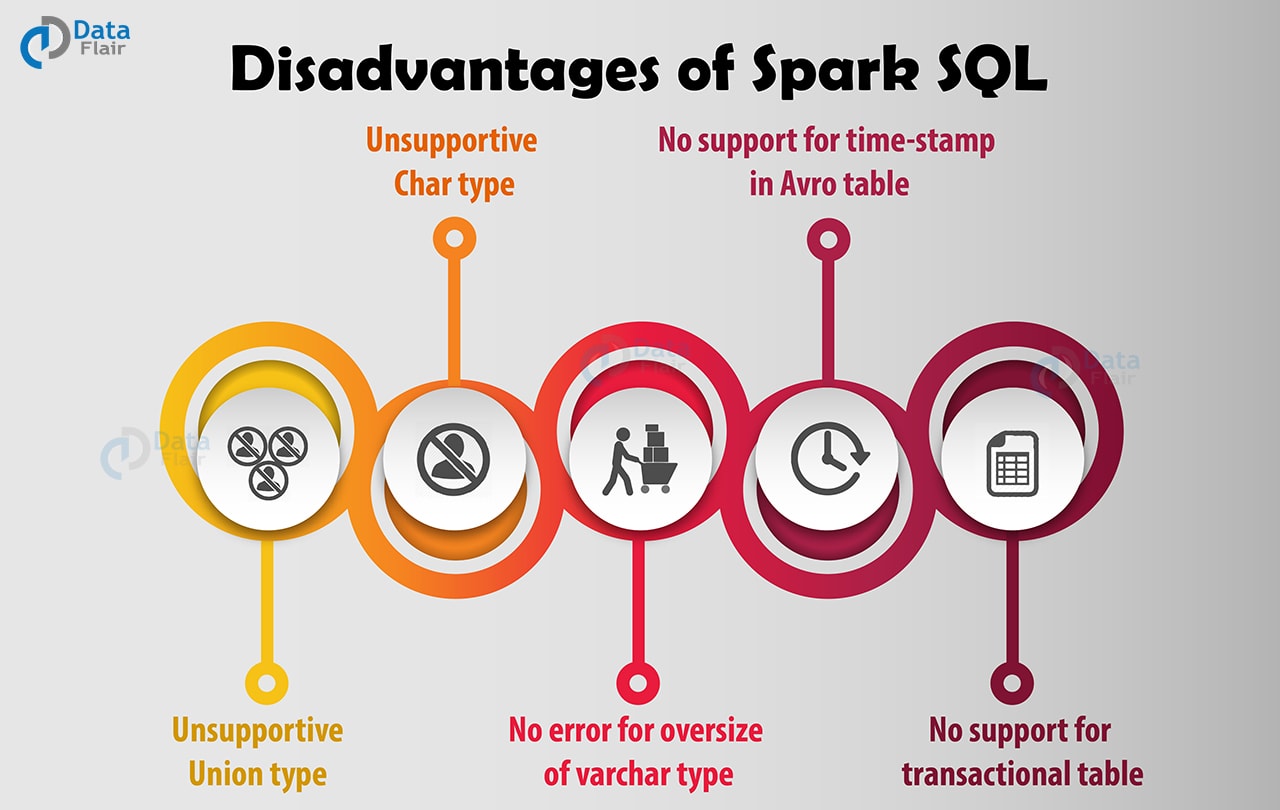
### vii. Advantages of Spark SQL



In this section of, we will discuss various advantages of Apache Spark SQL-

**a. Integrated**  
Apache Spark SQL mixes SQL queries with Spark programs. With the help of Spark SQL, we can query structured data as a distributed dataset (RDD). We can run SQL queries alongside complex analytic algorithms using tight integration property of Spark SQL.  
**b. Unified Data Access**  
Using Spark SQL, we can load and query data from different sources. The Schema-RDDs lets single interface to productively work structured data. For example, Apache Hive tables, parquet files, and JSON files.  
**c. High compatibility**  
In Apache Spark SQL, we can run unmodified Hive queries on existing warehouses. It allows full compatibility with existing Hive data, queries and UDFs, by using the Hive fronted and MetaStore.  
**d. Standard Connectivity**  
It can connect through JDBC or ODBC. It includes server mode with industry standard JDBC and ODBC connectivity.  
**e. Scalability**  
To support mid-query fault tolerance and large jobs, it takes advantage of RDD model. It uses the same engine for interactive and long queries.  
**f. Performance Optimization**  
The query optimization engine in Spark SQL converts each SQL query to a logical plan. Further, it converts to many physical execution plans. Among the entire plan, it selects the most optimal physical plan for execution. Read more about [Apache Spark performance tuning techniques](http://data-flair.training/blogs/apache-spark-performance-tuning/) in detail.  
**g. For batch processing of Hive tables**  
We can make use of Spark SQL for fast batch processing of Hive tables.

### viii. Disadvantages of Spark SQL



Apart from features, there are also some disadvantages of Spark SQL. Some of them are listed below-  
**a. Unsupportive Union type**  
Using Spark SQL, we cannot create or read a table containing union fields.  
**b. No error for oversize of varchar type**  
Even if the inserted value exceeds the size limit, no error will occur. The same data will truncate if read from Hive but not if read from Spark. SparkSQL will consider varchar as a string, meaning there is no size limit.  
**c. No support for transactional table**  
Hive transactions are not supported by Spark SQL.  
**d. Unsupportive Char type**  
Char type (fixed-length strings) are not supported. Like the union, we cannot read or create a table with such fields.  
**e. No support for time-stamp in Avro table.**

So, this was all in Spark SQL Tutorial. Hope you like our explanation.