

1. Apache CarbonData



1.1 Apache CarbonData Nedir?

Apache CarbonData, Apache Hadoop ekosisteminin ücretsiz ve açık kaynaklı, sütun odaklı bir veri depolama biçimidir. Hadoop'ta bulunan diğer sütunlu depolama dosya biçimlerine (RCFile ve ORC) benzer. Hadoop ortamındaki veri işleme çerçevelerinin çoğuyla uyumludur.

Apache CarbonData, büyük veri platformlarında (örneğin Apache Hadoop, Apache Spark, vb.) hızlı analizler yapmak için indeksli **sütunlu** bir veri biçimidir.

CarbonData, 2013 yılında Huawei'de geliştirildi. Proje, 2015 yılında Apache Topluluğu'na bağışlandı ve Haziran 2016'da Apache Kuluçka Merkezi'ne sunuldu. Proje, BlackDuck 2016 Yılın Açık Kaynak Çaylakları Büyük Veri kategorisinde en yüksek ödülü kazandı.

Çok seviyeli indeksleme, sıkıştırma ve kodlama teknikleri kullanarak performansı artırır. Bu sayede **petabaytlarca** veri üzerinde hızlı sorgulamalar yapılabilir.

CarbonData, Spark SQL ile uyumludur ve Spark SQL üzerinde gerçekleştirilen SQL sorgu işlemlerini destekler.

1.2 Neden Apache CarbonData?

Apache CarbonData, büyük veri analitiği alanında önemlidir çünkü veriyi verimli bir şekilde depolayarak ve sorgulayarak büyük hacimli veriler üzerinde hızlı analiz imkânı sunar. İşte neden önemli olduğuna dair bazı başlıca sebepler:

- Sütun Tabanlı Veri Depolama
- Hızlı sorgu yürütme
- Veri tutarlılığı
- Çok seviyeli indeksleme
- Yüksek ölçeklenebilirlik
- Gelişmiş sıkıştırma ve kodlama
- Çoklu veri depolama biçimleri (CSV, TSV, JSON, AVRO, Parquet, ORC ve diğerleri gibi birden fazla ham veri türünü destekler.)

- Büyük veri ekosistemiyle uyumlu (Hadoop ve Spark gibi yaygın büyük veri araçlarıyla uyumlu çalışır. Bu sayede, büyük veri işleme işlemleri için var olan altyapılara kolayca entegre edilebilir ve çeşitli analitik ihtiyaçlara esneklikle uyum sağlar.)
- Realtime + History Data (Gerçek zamanlı ve geçmiş verilerinin aynı tabloya yerleştirilmesine izin verir)
- Spark ile derin entegrasyon

1.3 Kullanım Alanları

CarbonData, çeşitli analitik iş yüklerinde faydalıdır. CarbonData'nın en sık kullanıldığı bazı tipik kullanım alanları aşağıda belgelenmiştir.

CarbonData şu alanlarda kullanılmaktadır (**ancak bunlarla sınırlı değildir**):

- **Banka Sektörü**
 - Dolandırıcılık tespit analizi
 - Risk profili analizi
 - Müşterilerin günlük bakiyelerini güncellemek için zip tablosu olarak kullanımı
- **Telekomünikasyon**
 - VIP müşteriler için sinyal anomalilerinin tespitiyle müşteri deneyimini iyileştirme
 - GSM verilerindeki MR ve CHR kayıtlarının analizi ile belirli bir zaman diliminde kule yükünü belirleme ve kule yapılandırmasını dengeleme
 - Erişim siteleri, video, ekran boyutu, yayın bant genişliği, kalite analizi ile ağ kalitesini ve yönlendirme yapılandırmasını belirleme
- **Web/İnternet**
 - Erişilen sayfa veya videoların, sunucu yüklerinin, yayın kalitesinin ve ekran boyutlarının analizi
- **Akıllı Şehir**
 - Araç takibi analizi
 - Alışılmadık davranışların analizi

Bu kullanım alanları genel olarak aşağıdaki kategorilere ayrılabilir:

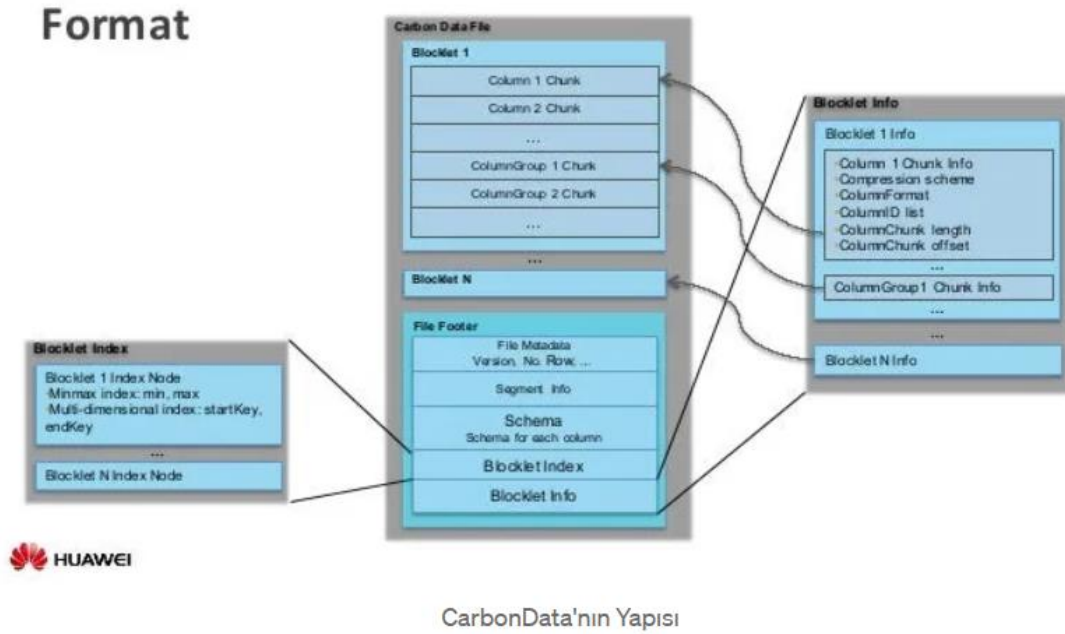
- Tam Tarama/Detaylı/Etkileşimli Sorgular
- Toplama/OLAP BI Sorguları
- Gerçek Zamanlı Veri Alma (Akış) ve Sorgular

1.4 CarbonData Özellikleri ve İşlevleri

Tablo Yönetimi

- **DDL (Create, Alter, Drop, CTAS):** CarbonData, kendi DDL yapısını sunarak CarbonData tablolarını oluşturma ve yönetme imkanı sağlar. Bu DDL, Hive ve Spark SQL formatına uygundur ve CarbonData işlevlerinden yararlanmak için ek özellikler ve yapılandırmalar sunar.
- **DML (Yükleme, Ekleme):** CarbonData, CarbonData tablolarındaki veriyi yönetmek için kendi DML yapısını sağlar. Yapılandırmalar aracılığıyla kullanıcı ihtiyaçlarına göre tamamen özelleştirilebilir.
- **Güncelleme ve Silme:** CarbonData, Büyük Veri üzerinde güncelleme ve silme işlemlerini destekler. CarbonData tablolarında IUD (Insert, Update, Delete) işlemlerini desteklemek için Hive'a benzer bir söz dizimi sağlar.
- **Segment Yönetimi:** CarbonData, CarbonData tablolarına aşamalı yüklemeleri etkili bir şekilde yönetmek için segment kavramını sunar. Segment yönetimi, tabloyu kolayca kontrol etmeyi, kolay saklama işlemlerini ve gerçekleştirilen işlemler için işlem yeteneği sağlamayı kolaylaştırır.
- **Bölümleme**
- **Sıkıştırma:** CarbonData, artan segment sayısını sıkıştırmak ve sorgu filtrelemelerini iyileştirmek için segmentleri sıkıştırma özelliğine sahiptir.
- **Harici Tablolar:** CarbonData, herhangi bir carbondata dosyasını okuyabilir, dosyadan otomatik olarak şema çıkarabilir ve Spark veya başka herhangi bir uygulama ile SQL sorguları gerçekleştirmek için ilişkisel bir tablo görünümü sağlayabilir.

1.5 Apache CarbonData'nın Yapısı



1.5.1 Çok Katmanlı Yapı:

Apache CarbonData, tablo, segment, blok ve sayfa düzeylerini içeren birden fazla katmanda yapılandırılmıştır. Bu hiyerarşik yapı, sorgu yürütme sırasında alakasız verileri atlayarak verimli veri alımına olanak tanır.

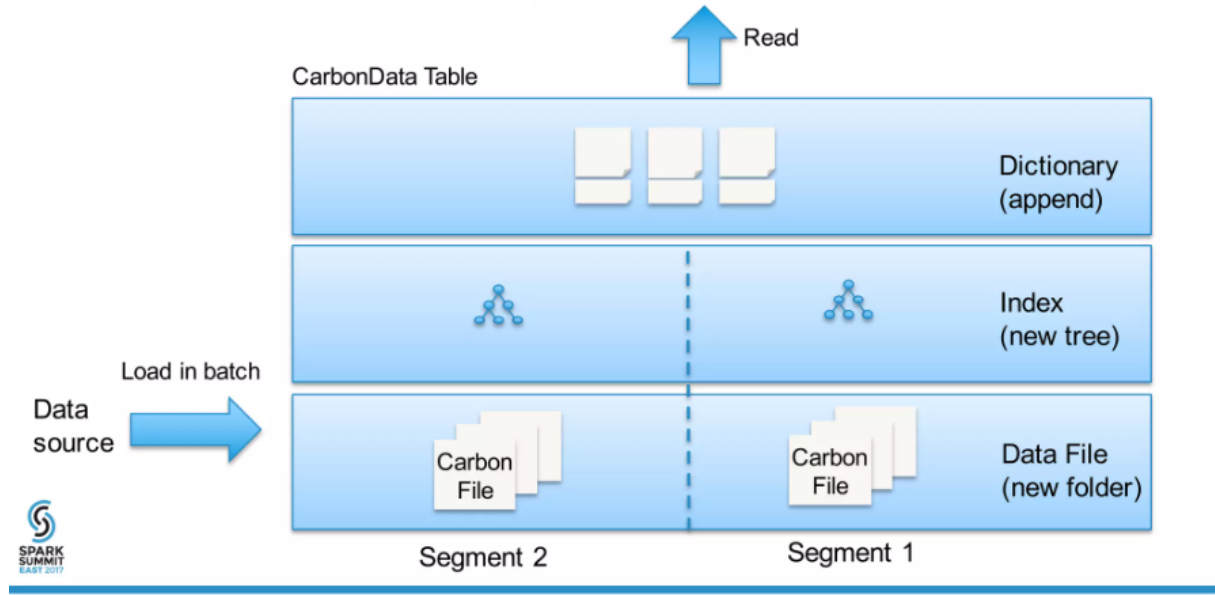
Tablo: Tablo, segmentlerin bir koleksiyonudur ve her segment bir veri dosyaları kümesini temsil eder.

Segment: Bir segment, her biri önemli miktarda veri depolayabilen birden fazla veri bloğu içerir.

Blok: Bir blok, blokletlere bölünür. Her bloklet, sütun biçiminde düzenlenmiş bir dizi sütun sayfası içerir.

Sayfa: Sayfa düzeyi gerçek verilerin depolandığı yerdir. Bu sayfalardaki veriler kodlanır ve sıkıştırılır, bu da veri alımını verimli hale getirir.

1.5.2 CarbonData Tablo Yapısı



1. **Genel Yapı:** CarbonData tablosu, verileri parçalar (segmentler) halinde saklar. Her bir segment, verilerin verimli bir şekilde okunup yazılmasını sağlar.
2. **Segmentler:** Şekilde Segment 1 ve Segment 2 olmak üzere iki segment bulunmaktadır. Bu segmentler, veriler batch (toplu) olarak yüklendiğinde oluşur. Segmentler, yeni veri geldikçe güncellenir ve yeni dosyalarla genişler.
3. **Data File (Veri Dosyası):** Her segmentin altında "Data File" (Veri Dosyası) adı verilen bir katman bulunur. Bu dosyalar, CarbonData formatında depolanan verilerin fiziksel olarak saklandığı yerlerdir. Yeni veri geldiğinde yeni klasörler (new folder) eklenir.
4. **Index (Dizin):** Veri dosyalarının üstünde bir "Index" (Dizin) katmanı yer alır. Bu dizinler, veriye hızlı erişim sağlamak için bir ağaç yapısında düzenlenmiştir ve her yeni segment için yeni bir ağaç yapısı oluşturulur. Diziner sayesinde, belirli veriler hızlıca bulunabilir.
5. **Dictionary (Sözlük):** En üstte "Dictionary" (Sözlük) katmanı vardır. Bu katman, tabloya eklenen verilerin değerlerini içeren bir yapıdır ve bu sayede veriler daha kompakt bir şekilde depolanır. Yeni veriler geldikçe sözlük yapısı güncellenir (append).

Özetle, CarbonData veriyi segmentlere bölerek depolar, bu segmentler indeks ve sözlük yapılarıyla desteklenir.

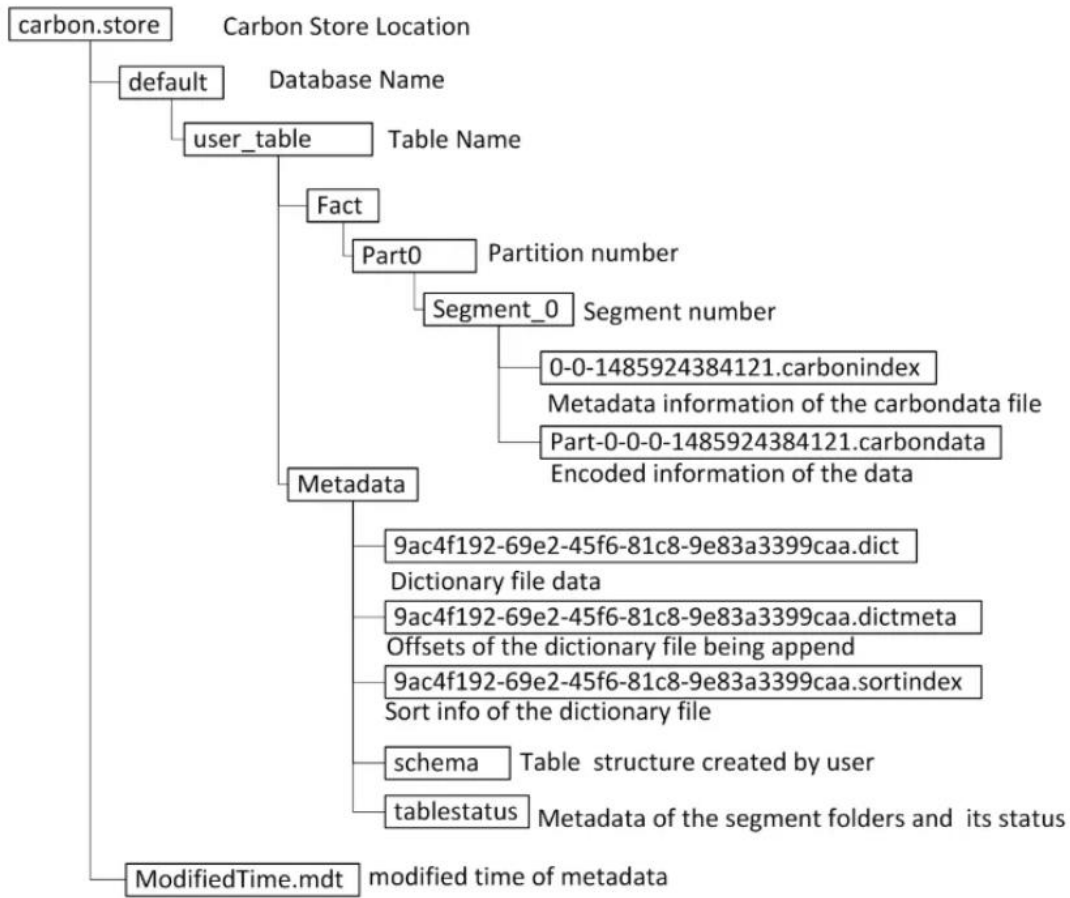
1.5.3 CarbonData Dosya Biçimi Mimarisi

Basitçe söylemek gerekirse, üç bileşeni vardır: dosya başlığı, dosya alt bilgisi ve bloklar.

Dosya alt bilgisi burada bir bakıma önemlidir. CarbonData, hızlı sorgu işleme ve optimize edilmiş taramalar elde etmek için bunu kullanır, çünkü bellekte okunur.

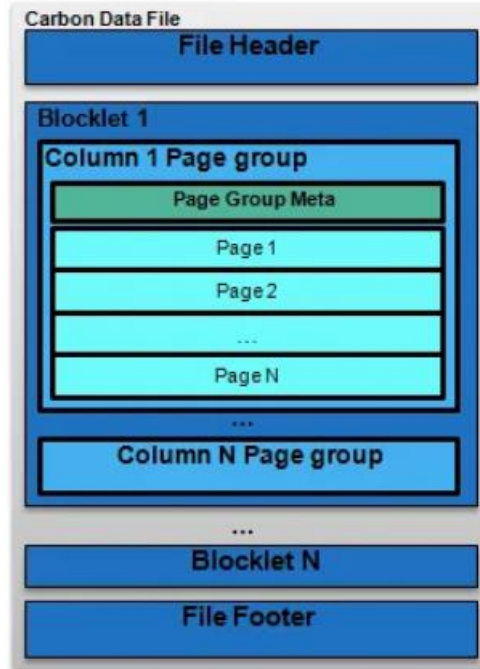
CarbonData dosyaları carbon.storelocation yapılandırmasıyla belirtilen konumda saklanır (carbon.properties'de yapılandırılır; yapılandırılmamışsa varsayılan ../carbon.store'dur).

Belgelerden alınan yapı şu şekildedir:



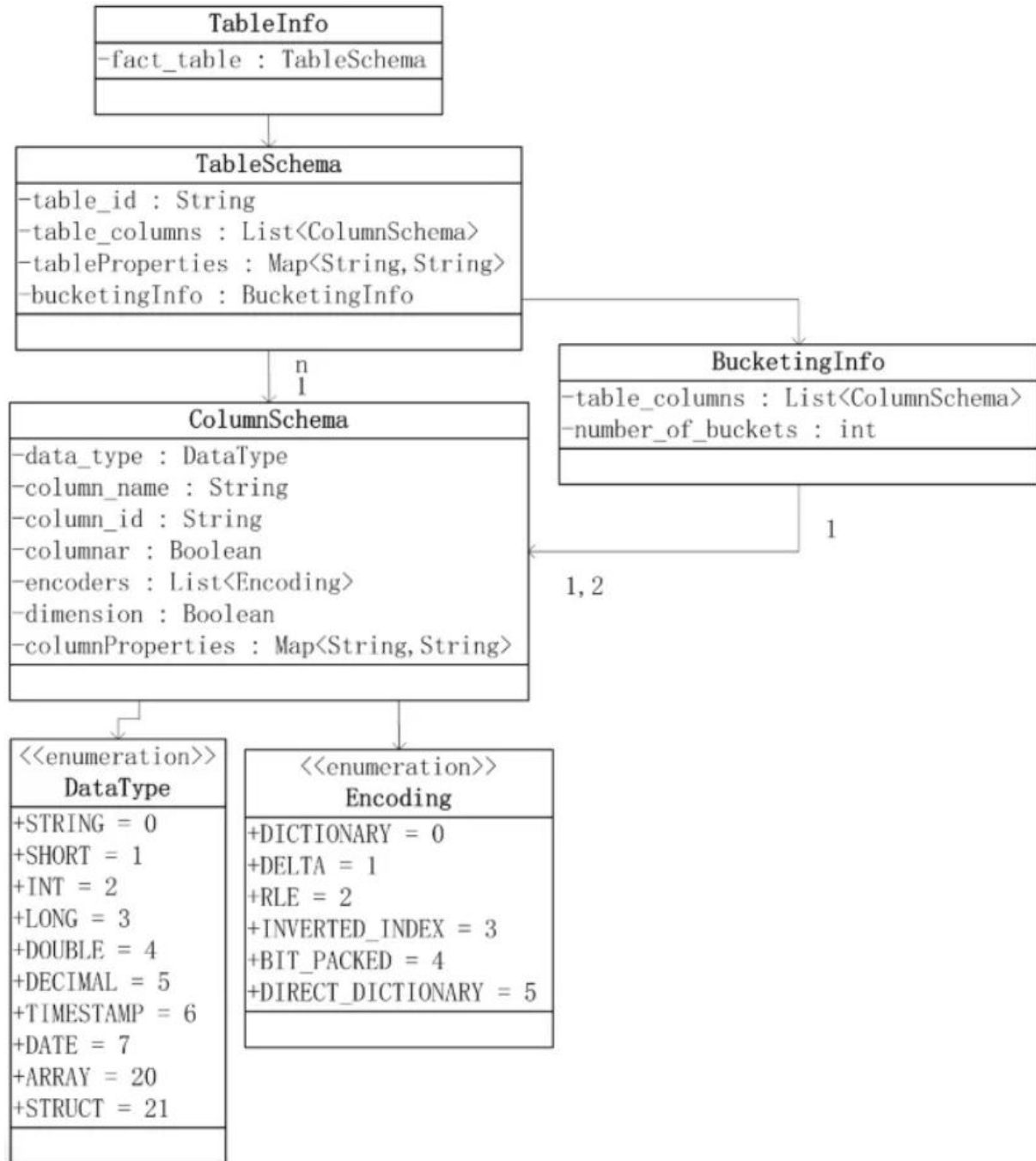
1. **ModifiedTime.htmlt**, dosyanın değişiklik zamanı özniteliğiyle meta verilerin zaman damgasını kaydeder. Drop table ve create table kullanıldığında, dosyanın değişiklik zamanı güncellenir. Bu, tüm veritabanları için ortaktır ve bu nedenle veritabanlarıyla paralel tutulur.
2. Varsayılan değer **veritabanı adıdır ve kullanıcı tablolarını** içerir. Kullanıcı herhangi bir veritabanı adı belirtmediğinde varsayılan değer kullanılır; aksi takdirde kullanıcı tarafından yapılandırılan veritabanı adı dizin adı olur. `user_table` ise tablo adıdır.
3. **Meta veri dizini** şema dosyalarını, `tablestatus` ve sözlük dosyalarını (`.dict`, `.dictmeta` ve `.sortindex` dahil) depolar. Üç tür meta veri bilgi dosyası vardır.
4. Veri ve dizin dosyaları **Fact adlı dizin** altında saklanır. Fact dizininde `Part0` bölüm dizini vardır, burada 0 bölüm numarasıdır.

5. Part0 dizininin altında Segment_0 dizini vardır ve burada 0 segment numarasını ifade eder.
6. Segment_0 dizininde carbondata ve carbonindex olmak üzere iki tür dosya vardır.



CarbonData dosya biçimi

Artık dosya formatını öğrendiğimize göre, Carbondata'nın kullanıcı verilerinin şemasını nasıl sakladığını tartışalım.



Yukarıdaki diyagramın açıklamasına geçmeden önce, "bucket table" teriminin ne anlama geldiğini anlamalıyız.

Bucketing, verilerin birçok dosya arasında eşit bir şekilde dağıtılmasını sağlayan bir yöntemdir. Her bir bucket için veriler, HDFS'teki tablo dizini altında ayrı bir HDFS dosyasında saklanır. Bucketing, partitioning (bölümlendirme) alternatifidir. Veri setlerini daha yönetilebilir parçalara ayırır.

Şimdi, şematik diyagramı inceleyelim:

TableSchema sınıfı: TableSchema sınıfı, tablo adını saklamaz; bu ad, dizin adından (user_table) çıkarılır. tableProperties, tablo ile ilgili özellikleri kaydetmek için kullanılır, örneğin: table_blocksize.

ColumnSchema sınıfı: Kodlayıcılar, sütun depolama işlemlerinde kullanılan kodlamayı kaydetmek için kullanılır. columnProperties, sütun ile ilgili özellikleri kaydetmek için kullanılır.

BucketingInfo sınıfı: Bir bucket tablosu oluştururken, tabloda kaç tane bucket olacağını ve bucket'ların hangi sütuna göre bölüneceğini belirtebilirsiniz.

DataType sınıfı: CarbonData tarafından desteklenen veri türlerini tanımlar.

Encoding sınıfı: CarbonData dosyalarında kullanılabilecek çeşitli kodlamaları tanımlar.

Şimdiye kadar her şey, CarbonData'nın verilerinizle ilgili bilgileri nasıl sakladığı hakkında idi. Şimdi, CarbonData'nın verilerimizi nasıl sakladığını inceleyeceğiz.

1.5.4 Çok Katmanlı İndeksleme Yapısı

CarbonData, çok katmanlı indeksleme (multi-level indexing) ile veri erişim hızını artırır. Min-Max İndeksleme, Bloom Filtresi, Materialized View, İkincil İndeks gibi teknikler, sorgu süresini minimuma indirir.

Min-Max İndeks

- Min-Max indeksleme, her segment için minimum ve maksimum veri değerlerini kaydeder. Bu indeksleme, sorgulama sırasında yalnızca ilgili veri aralığında kalan segmentlerin taranmasına izin verir.
- Örneğin, belirli bir tarih aralığında veri arıyorsanız, Min-Max indeksler yalnızca o aralığa denk gelen segmentleri tarayarak sorgu süresini azaltır.
- **Avantajı:** Veriyi hızlıca budayarak, I/O işlemlerini ve tarama maliyetlerini azaltır.

Secondary İndeks (İkincil İndeks)

- İkincil indeksler, belirli kolonlarda veri arama işlemini hızlandırmak için kullanılır. Bu indeksler, esas tablodan ayrı olarak belirli sütunlar üzerinde indeks oluşturur.
- İkincil indeksler, özellikle birden fazla koşula dayalı sorgularda faydalıdır. Örneğin, kullanıcıya göre filtreleme veya ürün türüne göre gruplama gibi durumlarda veriye daha hızlı erişim sağlar.
- **Avantajı:** Sık sorgulanan belirli kolonlar üzerinde veriye daha hızlı erişim sağlar.

Materialized View (Materyalize Görünüm)

- Materialized View, verilerin özetlerini veya önceden hesaplanmış sonuçlarını saklar. Örneğin, belirli kolonlar arasında toplama veya gruplama işlemleri önceden yapılarak bir tabloya kaydedilir.
- Bu önceden işlenmiş veriler, tekrar eden sorgularda performansı büyük ölçüde artırır çünkü ham veriyi yeniden işlemek yerine hazır sonuçlardan faydalanılır.
- **Avantajı:** Yoğun işlem gerektiren sorgularda (toplam, ortalama hesaplama gibi) veri erişimini hızlandırarak sorgulama süresini ciddi ölçüde azaltır.

1.6 Apache CarbonData ve Apache Spark Entegrasyonu

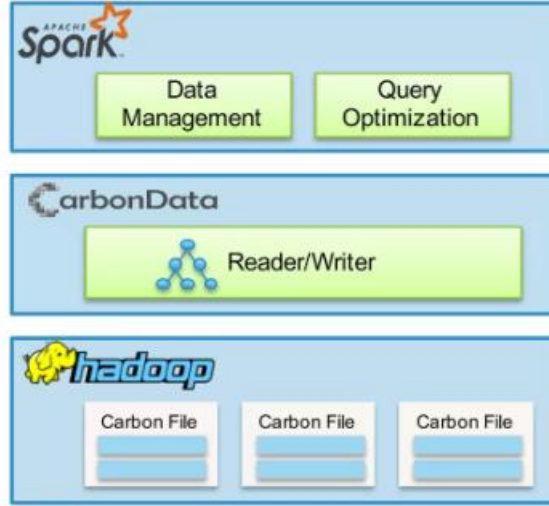
Apache CarbonData, büyük veri analitiği için yüksek performanslı bir veri yönetimi çözümüdür ve Apache Spark ile entegre çalışarak modern veri işleme ihtiyaçlarına güçlü bir yanıt sunar.

Apache Spark, veri işleme, makine öğrenimi, veri akışı ve grafik işleme gibi alanlarda büyük veri kümelerini hızla işleyebilme yeteneğiyle bilinen bir analiz platformudur.

CarbonData ise kolon bazlı ve sıkıştırılmış veri saklama yapısı ile veri depolama süreçlerini optimize eder. Bu iki güçlü teknoloji bir araya geldiğinde, büyük veri projelerinde veri saklama ve analiz işlemlerinde güzel bir performans artışı sağlanır.

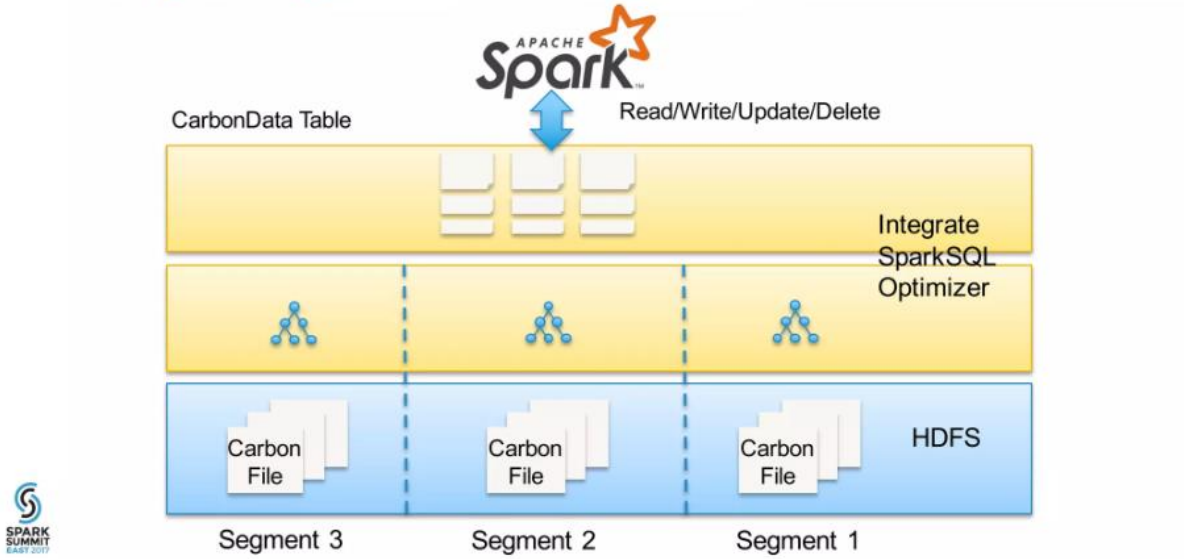
Apache CarbonData'nın Spark ile entegrasyonu, Spark SQL üzerinden veri okuma, yazma, güncelleme ve silme işlemlerini mümkün kılar.

Spark'ın dağıtık veri işleme gücü, CarbonData tablolarında yapılan işlemlerin hızını artırır ve daha etkin bir analiz süreci sunar. Ayrıca, CarbonData'nın indeksleme ve veri sıkıştırma özellikleri sayesinde büyük veri kümeleri üzerinde yapılan sorgular daha hızlı ve verimli hale gelir.



1. Üst kısımda Spark bileşenleri (Data Management ve Query Optimization) bulunur. Spark, veri yönetimi ve sorgu optimizasyonunu sağlar.
2. Orta katmanda CarbonData'nın "Reader/Writer" (Okuyucu/Yazıcı) bileşeni yer alır. Bu bileşen, Spark ile veri yazma ve okuma işlemlerini yönetir.
3. Alt katmanda Hadoop bileşeni ve CarbonData'nın dosya yapısı (Carbon File) gösterilmiştir. Veriler Hadoop üzerinde CarbonData dosyaları şeklinde depolanır.

Deep Integration with SparkSQL



1. CarbonData Tablosu ve Segmentler

- CarbonData tablosu, verileri segmentler halinde saklar. Görselde Segment 1, Segment 2 ve Segment 3 olarak gösterilen bu parçalar, verilerin farklı dosya yapılarında tutulduğu alanlardır.
- Her segment, kendi Carbon dosyalarını içerir. Bu segment yapısı, verilerin yönetimini ve erişimini daha verimli hale getirir.

2. HDFS (Hadoop Distributed File System)

- CarbonData dosyaları HDFS üzerinde depolanır. HDFS, büyük veri kümelerini dağıtık bir şekilde depolayarak veri işleme sürecini hızlandırır ve güvenilirliği artırır.

3. Veri İşleme Katmanları

- **En üst katman**, SparkSQL'in CarbonData tablo verisi ile etkileşimde olduğu yerdir. Bu katman, veri okuma/yazma işlemlerinin yapıldığı alandır.
- **Orta katmanda**, SparkSQL optimizasyon katmanı bulunur. Bu katman, CarbonData tablosundaki sorguların hızını artırır.
- **Alt katmanda** ise fiziksel veri dosyaları (Carbon File) bulunur. Bu dosyalar HDFS'de segmentlere göre saklanır.

1.7 Kurulum

Apache CarbonData'nın genel yapısından bahsettikten sonra kurulum nasıl yapılır adım adım gösterelim. Örnek kurulum Windows işletim sistemi üzerine gerçekleştirilecektir. Daha sonra yapılacak olan örnek uygulama için spark-shell kullanılacaktır.

Apache CarbonData, Apache Spark ile entegre olarak çalıştığı için öncelikle Apache Spark'ın kurulu olması gerekir.

NOT: Apache CarbonData'nın doğru bir şekilde çalışabilmesi için mutlaka Apache Spark sürümüyle uyumlu olması gerekir. Apache Spark'ın doğru bir şekilde çalışabilmesi için de Java sürümünün uyumlu olması gerekir. Yani Apache Spark, Apache CarbonData ve Java sürümlerinin birbirleriyle uyumlu olması gerekir

Gereksinimler:

- Java 8 (Apache CarbonData ile daha verimli çalışır)
- Apache CarbonData sürümü ile uyumlu Apache Spark versiyonu

Kurulum yapılacak cihazımızda JAVA yüklü olup olmadığı kontrol edilir. Eğer cihazınız üzerinde java kurulu değilse aşağıda sayfadan indirilerek kurulum yapılabilir.

<https://www.oracle.com/java/technologies/downloads/#java8>

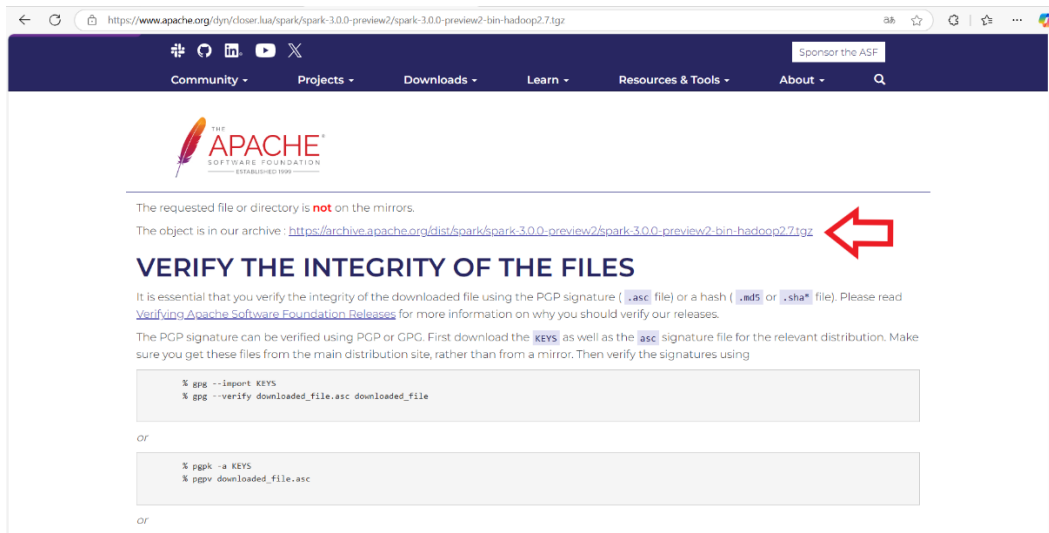
```
Command Prompt
Microsoft Windows [Version 10.0.17763.973]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\admin>java -version
java version "1.8.0_201"
Java(TM) SE Runtime Environment (build 1.8.0_201-b09)
Java HotSpot(TM) Client VM (build 25.201-b09, mixed mode)

C:\Users\admin>
```

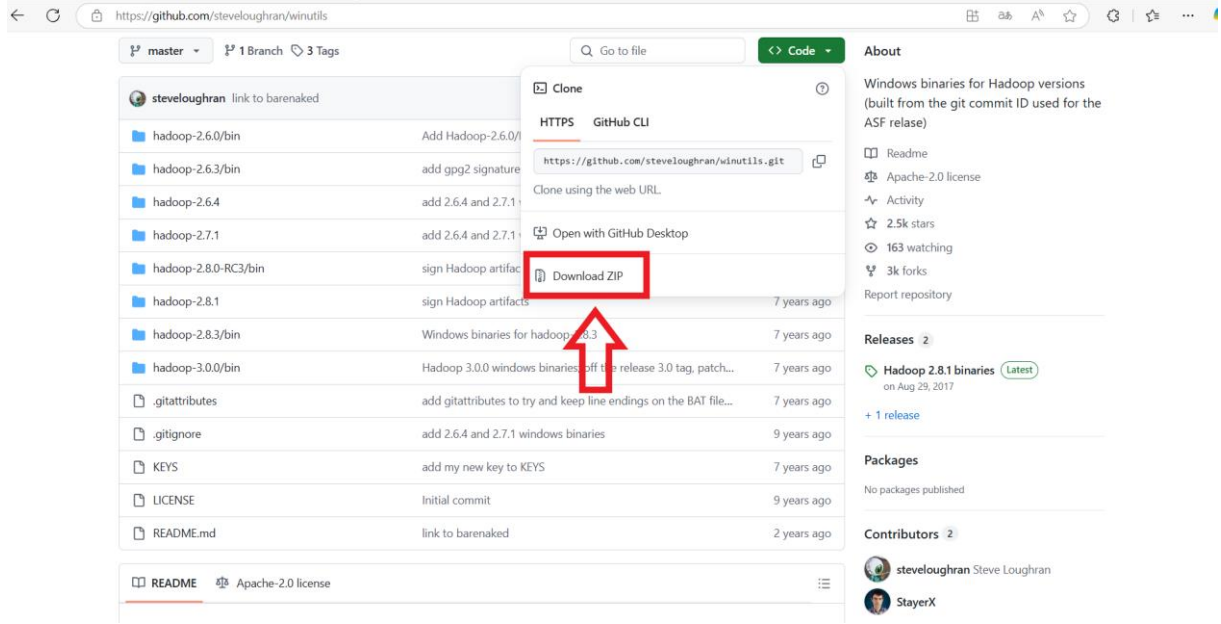
1.7.1 Apache Spark Kurulumu

Java kontrolünü yaptıktan sonra CarbonData ile uyumlu olan bir Apache Spark dosyası belirtilen adresten indirilir.



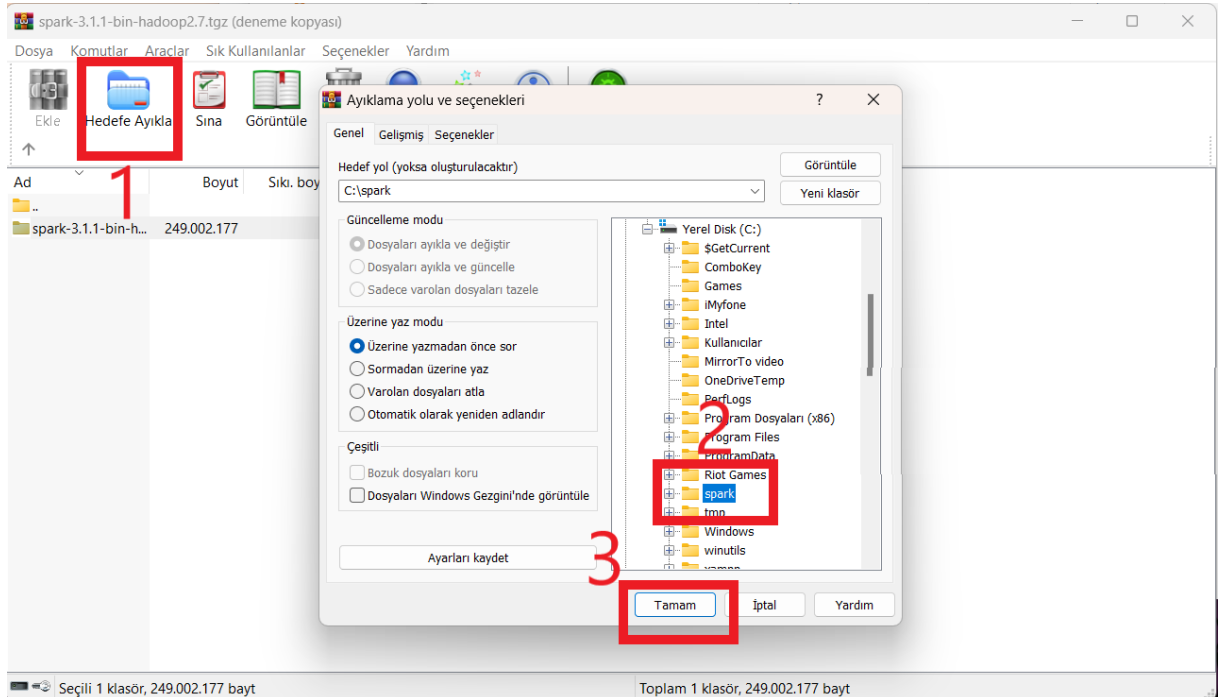
Apache Spark indirildikten sonra Windows ortamında Hadoop kullanabilmek için **winutils.exe** uygulaması indirilir. İlgili programı **Github** sayfasından elde edilebilir.

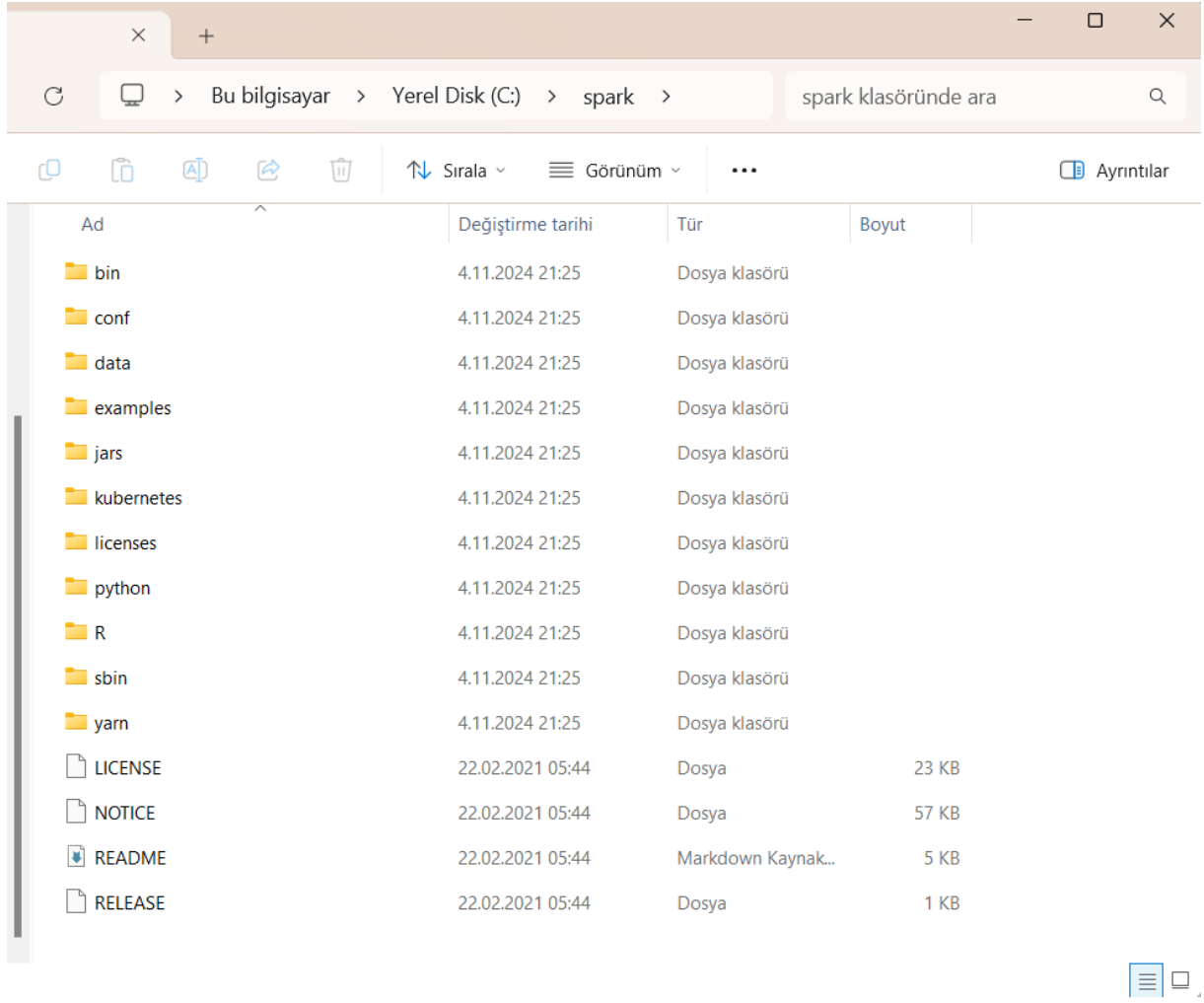
<https://github.com/steveloughran/winutils>



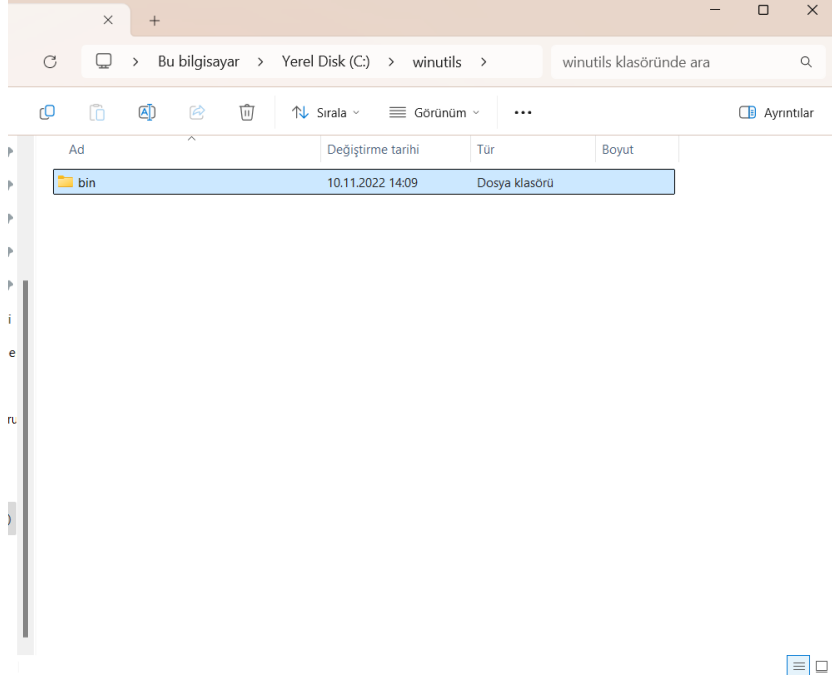
Tüm indirme işlemleri tamamlandıktan sonra kurulum kısmına gelinir.

İndirilen spark dosyası **Windows C:/** dizini altında **“spark”** diye klasör oluşturulur ve indirilen spark dosyası Winrar uygulaması ile açılır ve spark klasörüne ayıklanır (zip dosyasından çıkartılır).



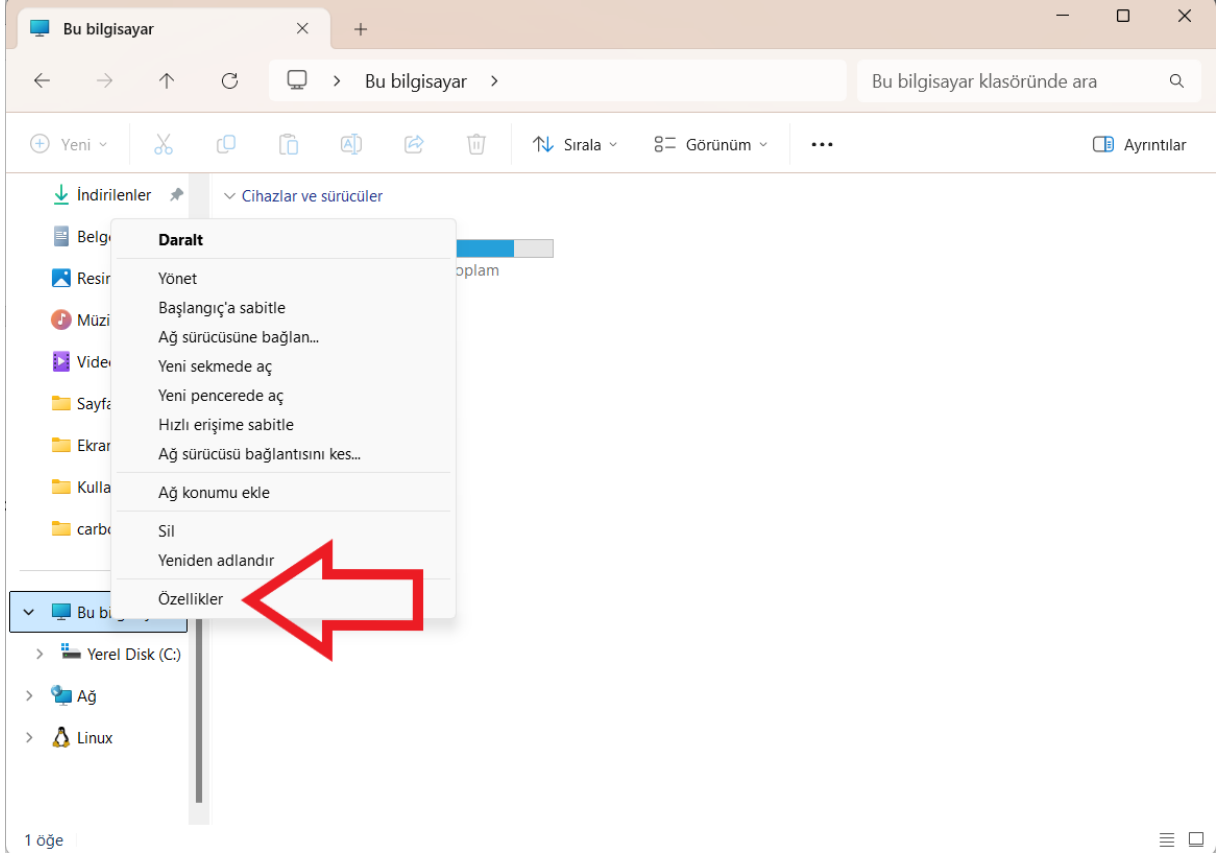


Daha sonra Hadoop kullanımı için indirilmiş olan winutils.exe dosyası **Windows C:/ dizini altında winutils klasörü oluşturularak** winutils.exe dosyası bu klasörün içine **kopyalanır**.

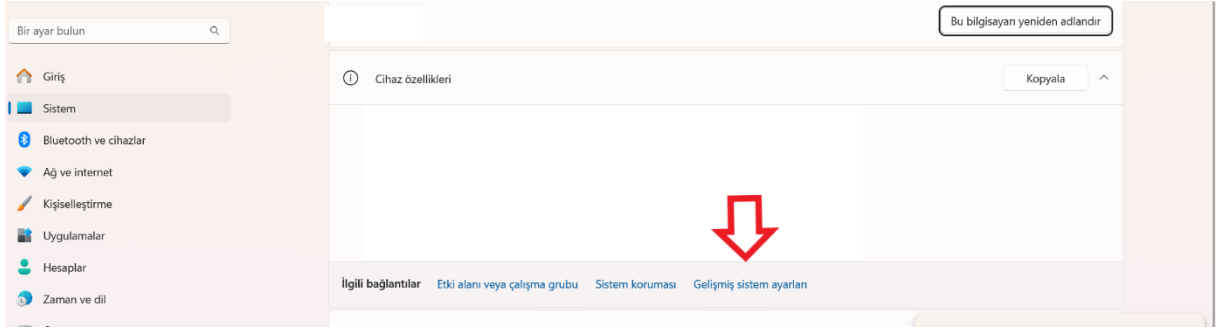


Tüm bu işlemler yapıldıktan sonra Windows da gelişmiş sistem ayarlamaları yapılır.

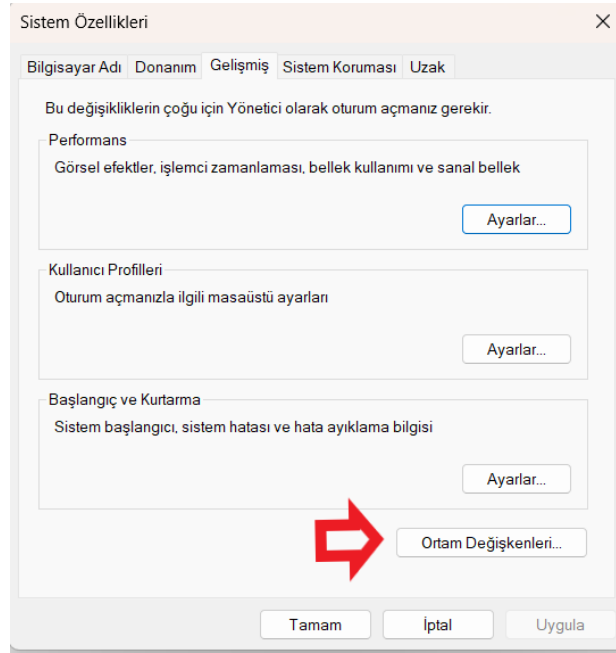
“Bu bilgisayara” sağ tıklanır ardından **“özelliklere”** tıklanır.



Ardından **“gelişmiş sistem ayarları”** kısmına girilir.



Ardından **“Ortam Değişkenler...”** kısmına girilir



Gerekli olan dosya yolu (path) girilir.

Yeni Kullanıcı Değişkeni

Değişken adı: JAVA_HOME

Değişken: C:\Program Files\Java\jdk1.8.0_231

Dizine Gözet... Dosyaya Gözet... Tamam İptal

Kullanıcı Değişkenini Düzenle

Değişken adı: SPARK_HOME

Değişken: C:\spark

Dizine Gözet... Dosyaya Gözet... Tamam İptal

Kullanıcı Değişkenini Düzenle

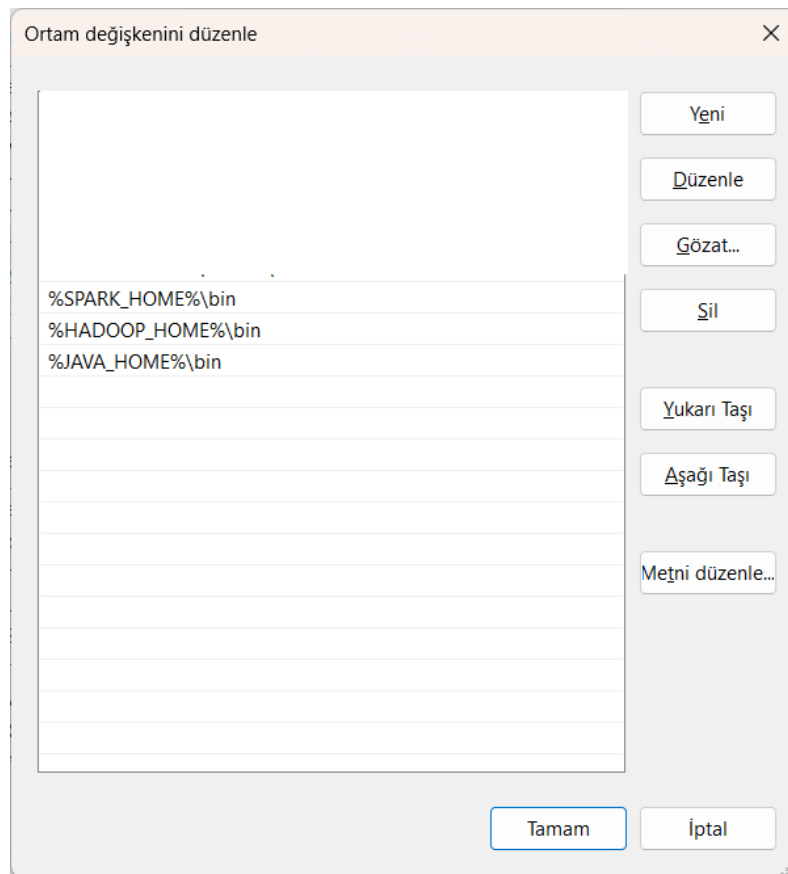
Değişken adı: HADOOP_HOME

Değişken: C:\winutils

Dizine Gözet... Dosyaya Gözet... Tamam İptal

Kurulu olan dosya dizinlerimizi ekledikten sonra path yapılandırma işlemi yapılmalıdır

Path'e tıkladıktan sonra düzenleye basılır ve 3 path eklenir.



Tüm bu yapılan değişikliklerin geçerli olması için **Windows işletim sistemi yeniden başlatılır** ve spark kurulumu tamamlanmış olur.

Spark'ın kurulduğunu görebilmek için **CMD(Komut istemi)** açılır ve **pyspark** komutu yazılır. Spark çıktısı alındığında spark'ın başarılı bir şekilde kurulduğu öğrenilir.

```
C:\Users\umut>pyspark
Python 3.10.7 (tags/v3.10.7:6cc6b13, Sep  5 2022, 14:08:36) [MSC v.1933 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
24/11/05 20:07:09 WARN Utils: Service 'SparkUI' could not bind on port 4040. Attempting port 4041.
Welcome to

  /---\
 /  V  \--\---\---\---\---\
/_--/_--\---\---\---\---\---\  version 3.1.1
/_--/_--\---\---\---\---\---\
/_--/_--\---\---\---\---\---\

Using Python version 3.10.7 (tags/v3.10.7:6cc6b13, Sep  5 2022 14:08:36)
Spark context Web UI available at http://172.18.252.50:4041
Spark context available as 'sc' (master = local[*], app id = local-1730826429759).
SparkSession available as 'spark'.
>>> 24/11/05 20:07:20 WARN ProcfMetricsGetter: Exception when trying to compute pagesize, as a result reporting of Proc
essTree metrics is stopped
```

Başarılı bir şekilde çalıştı.

Şimdi de CarbonData kullanarak Spark SQL komutları ile bir tablo oluşturalım.

```
spark.sql("""  
CREATE TABLE carbon_table (  
    id INT,  
    name STRING,  
    age INT  
) USING carbon  
""")
```



A screenshot of a terminal window titled "Komut İstemi - spark-shell". The prompt is "scala>". The command entered is "spark.sql(""" CREATE TABLE carbon_table (id INT, name STRING, age INT) USING carbon """)". The command is shown with vertical line indicators for each line.

Oluşturduğumuz bu tabloyu Spark SQL komutu ile görelim.

```
spark.sql("SHOW TABLES").show()
```



A screenshot of a terminal window titled "Komut İstemi - spark-shell". The prompt is "scala>". The command entered is "spark.sql("SHOW TABLES").show()". The output is displayed as a table with three columns: "database", "tableName", and "isTemporary". The output shows two rows: "default| carbon_table| false" and "default|example_table| false".

database	tableName	isTemporary
default	carbon_table	false
default	example_table	false

Oluşturduğumuz carbon_table tablosu başarılı bir şekilde gözükmetedir.

Şimdi de bu tabloya veri ekleyelim.

// Veri ekleme

```
spark.sql("""
```

```
INSERT INTO carbon_table VALUES
```

```
(1, 'Alice', 23),
```

```
(2, 'Bob', 34),
```

```
(3, 'Charlie', 29)
```

```
""")
```



```
Komut İstemi - spark-shell --i x + v

scala> // Veri ekleme

scala> spark.sql("""
| INSERT INTO carbon_table VALUES
|   (1, 'Alice', 23),
|   (2, 'Bob', 34),
|   (3, 'Charlie', 29)
| """)
```

Veri ekleme komutunu da başarılı bir şekilde çalıştırdık.

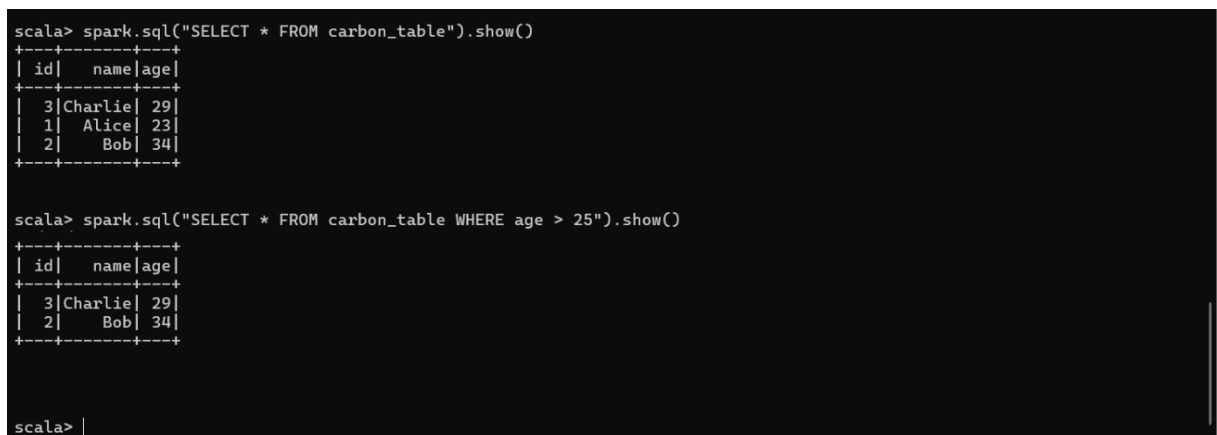
Şimdi de bu tablo üzerinde basit sorgulamalar yapalım.

(Tablodaki tüm verileri getirme).

```
spark.sql("SELECT * FROM carbon_table").show()
```

(Tablodaki yaşı 25'ten büyük olanları getirme).

```
spark.sql("SELECT * FROM carbon_table WHERE age > 25").show()
```



```
scala> spark.sql("SELECT * FROM carbon_table").show()
+-----+
| id|  name|age|
+-----+
|  3|Charlie| 29|
|  1|  Alice| 23|
|  2|   Bob| 34|
+-----+

scala> spark.sql("SELECT * FROM carbon_table WHERE age > 25").show()
+-----+
| id|  name|age|
+-----+
|  3|Charlie| 29|
|  2|   Bob| 34|
+-----+

scala> |
```

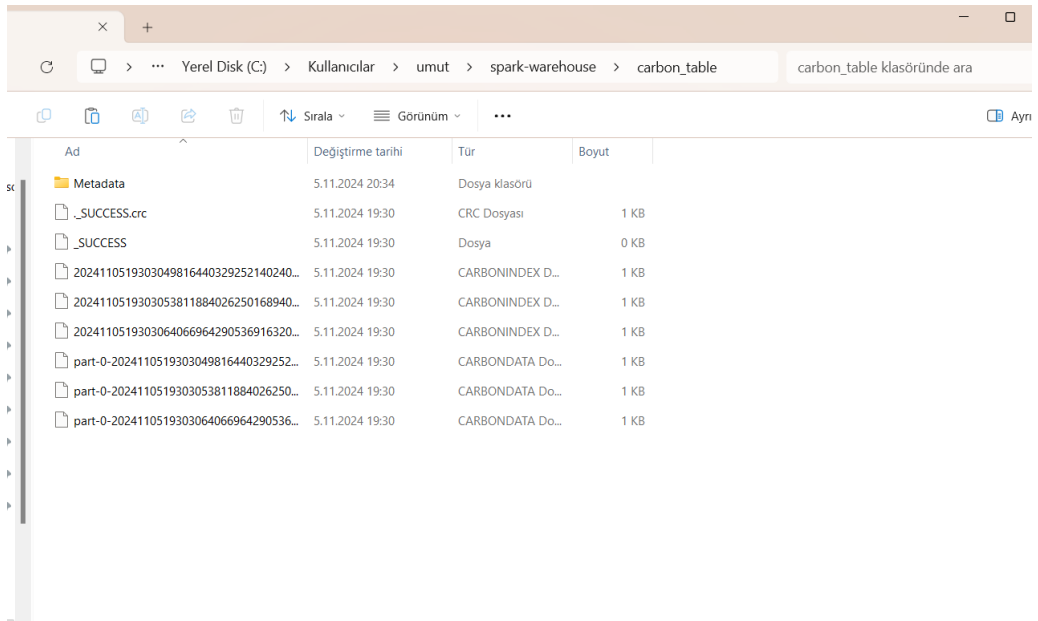
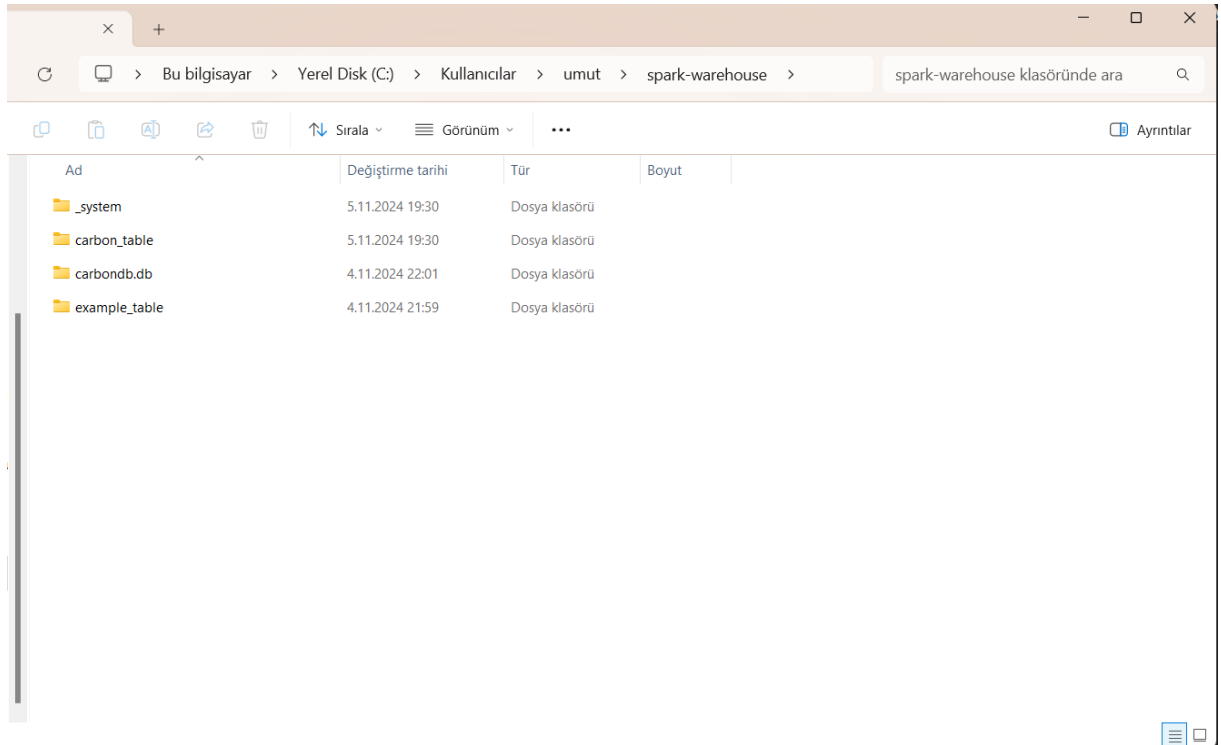
```
spark.sql("SELECT COUNT(*) FROM carbon_table").show()
```

```
scala> spark.sql("SELECT COUNT(*) FROM carbon_table").show()
+-----+
|count(1)|
+-----+
|      3|
+-----+

scala> |
```

Başarılı bir şekilde sorgulamalarımızı gerçekleştirdik.

Dosyalardan tablomuzun yoluna giderek oluştuğunu görebiliriz.



1.8 Neler Öğrendik?

1. Apache CarbonData Nedir Ne İş Yapar?
2. Apache CarbonData'nın Mimarisi
3. Apache CarbonData'nın Kurulumu
4. Apache CarbonData ile Örnek Uygulama

Dinlediğiniz İçin Teşekkürler.

Ad: Umut Sefkan

Soyad: SAK

Numara: 02210224071

Ders: Yazılım Mühendisliğinde Gelişmeler-3

Konu: Apache CarbonData