Databricks Community Edition
https://community.cloud.databricks.com/?o=676561995749089

第2周 Spark maks big data easy

Big Data引言

1. 大数据的特性：5V
    a. Volume
        i. 如何访问海量数据
    b. Velocity
        i. 新数据的生成速度
        ii. 数据移动的速度
        iii. 如何处理和分析：查询，实时报告
    c. Variety
        i. 不同的数据结构和来源
    d. Veracity真实性
        i. 数据质量和准确性
    e. Value
        i. 可共享，可实施，可视
2. 字节容量换算

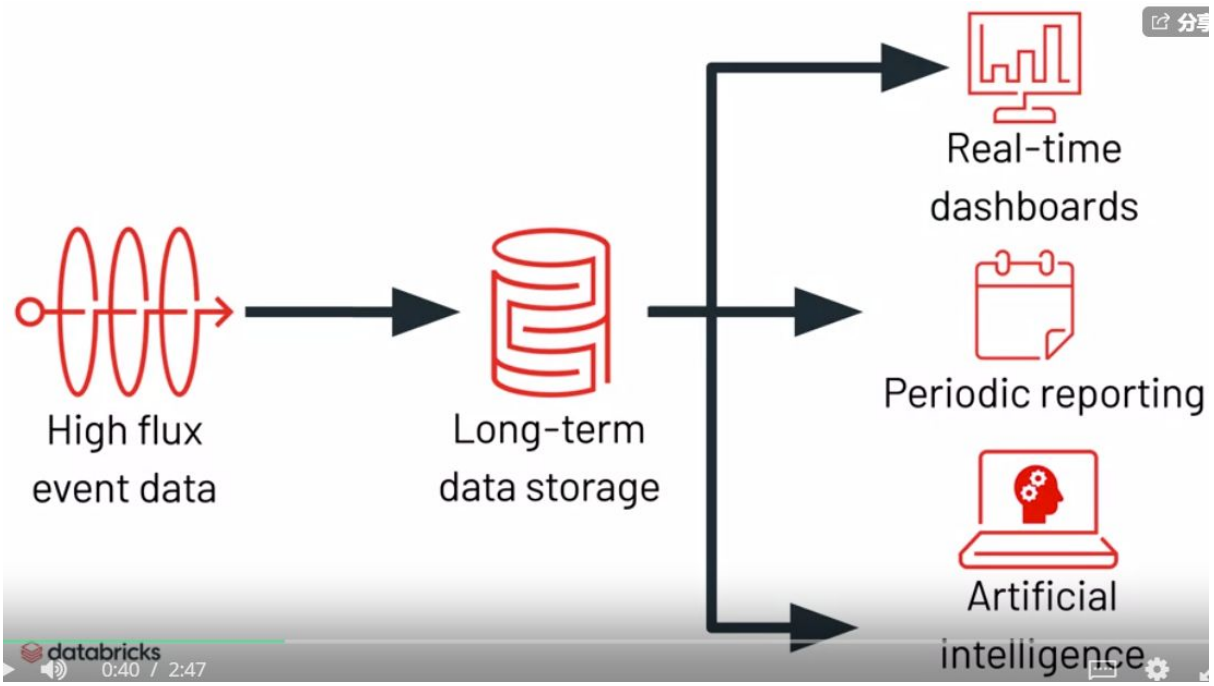
Multiple-byte units V • T • E

Decimal			Binary				
Value	Metric		Value	IEC		JEDEC	
1000	kB	kilobyte	1024	KiB	kibibyte	KB	kilobyte
1000^2	MB	megabyte	1024^2	MiB	mebibyte	MB	megabyte
1000^3	GB	gigabyte	1024^3	GiB	gibibyte	GB	gigabyte
1000^4	TB	terabyte	1024^4	TiB	tebibyte	–	
1000^5	PB	petabyte	1024^5	PiB	pebibyte	–	
1000^6	EB	exabyte	1024^6	EiB	exbibyte	–	
1000^7	ZB	zettabyte	1024^7	ZiB	zebibyte	–	
1000^8	YB	yottabyte	1024^8	YiB	yobibyte	–	

Orders of magnitude of data

3. 大数据带来的常见问题
    a. 缺少工具
        i. 扩展性不在公司蓝图中
        ii. 试图使用即时方法来解决大数据需求
        iii. 基础设施无力承载大数据
    b. 多个数据来源
        i. 使用各种各样的工具来访问数据
    c. 没有单一的真相来源

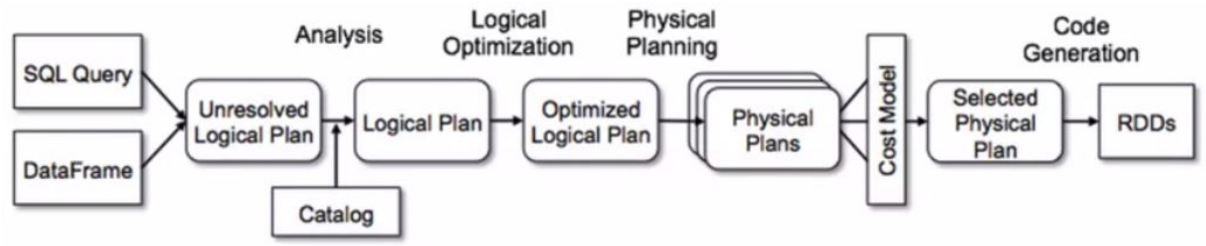
4. 大数据的需要



   a. 数据收集
   b. 数据存储
      i. 多个不同的数据库来满足特定的业务需求
      ii. 一个中心化的数据仓库
      iii. 数据湖（流行）
   c. 数据挖掘

Apache Spark(TM)引言

1. 使用Apache Spark处理大数据的好处
   a. Apache Spark
      i. 分布式计算引擎
      ii. 处理不同来源的数据和不同存储格式
      iii. 多种语言的API访问（Scala，Python，R，SQL）
   b. Spark SQL
      i. 一个用于结构化数据处理的Spark库
      ii. 允许我们使用SQL来访问Spark
      iii. 好处
         1. 使用简单
         2. 查询优化



第3周 在Databricks中使用Spark SQL

.dbc：可以import进Databricks的notebook文件集合

基本查询

- Temporary Views：临时视图

```
CREATE OR REPLACE TEMPORARY VIEW SSADistinctNames AS
  SELECT DISTINCT firstName AS ssaFirstName
  FROM SSANames;
```

- Parquet：一个开源的，基于列的文件格式

```
DROP TABLE IF EXISTS ssaNames;
CREATE TABLE ssaNames USING parquet OPTIONS (
  path "/mnt/training/ssn/names.parquet",
  header "true"
)
```

- 表连接

```
SELECT firstName
FROM PeopleDistinctNames
JOIN SSADistinctNames ON firstName = ssaFirstName
```

数据可视化

- 创建表

```
DROP TABLE IF EXISTS movieRatings;
CREATE TABLE movieRatings (
  userId INT,
  movieId INT,
  rating FLOAT,
  timeRecorded INT
) USING csv OPTIONS (
  PATH "/mnt/training/movies/20m/ratings.csv",
  header "true"
);
```

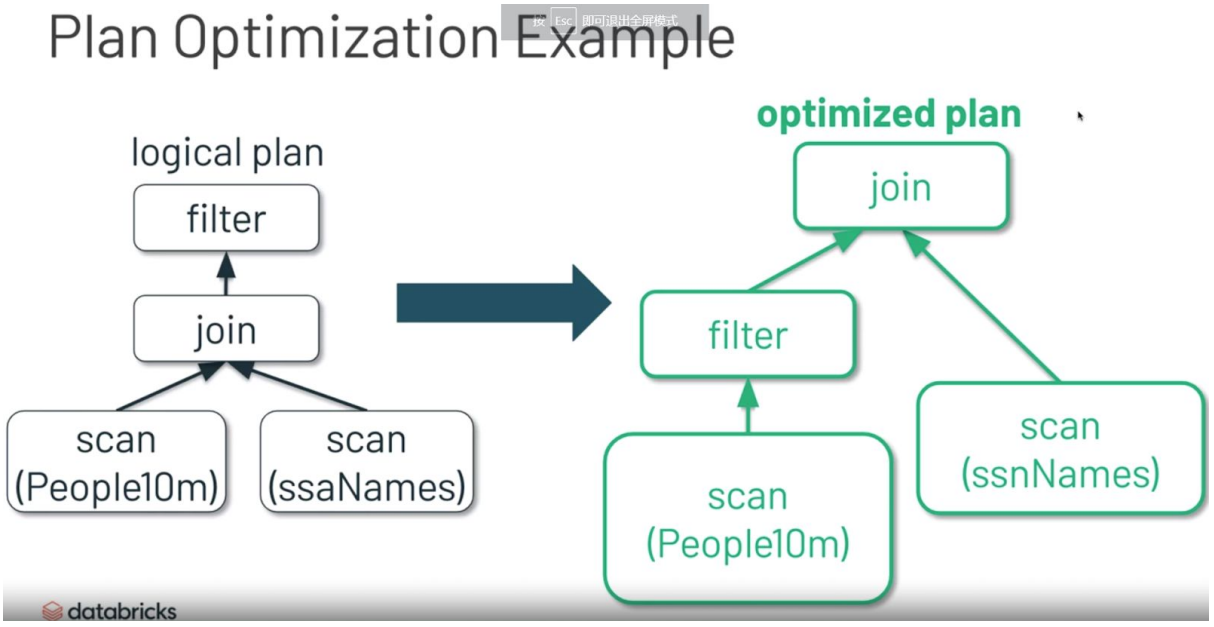
- 转换某列的数据类型

```
SELECT
  rating,
  CAST(timeRecorded as timestamp)
FROM
  movieRatings;
```

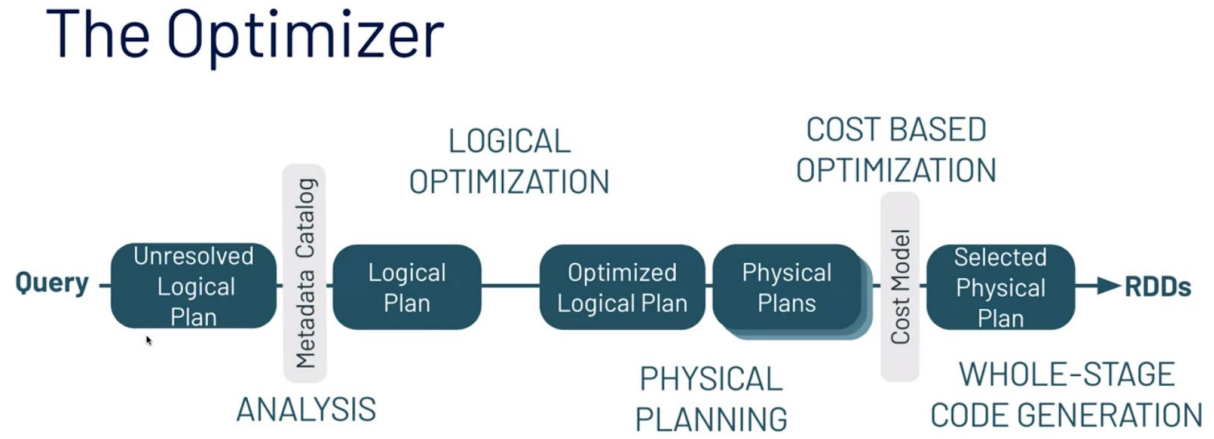
第4周 Spark Under the Hood

Spark SQL Powered Queries

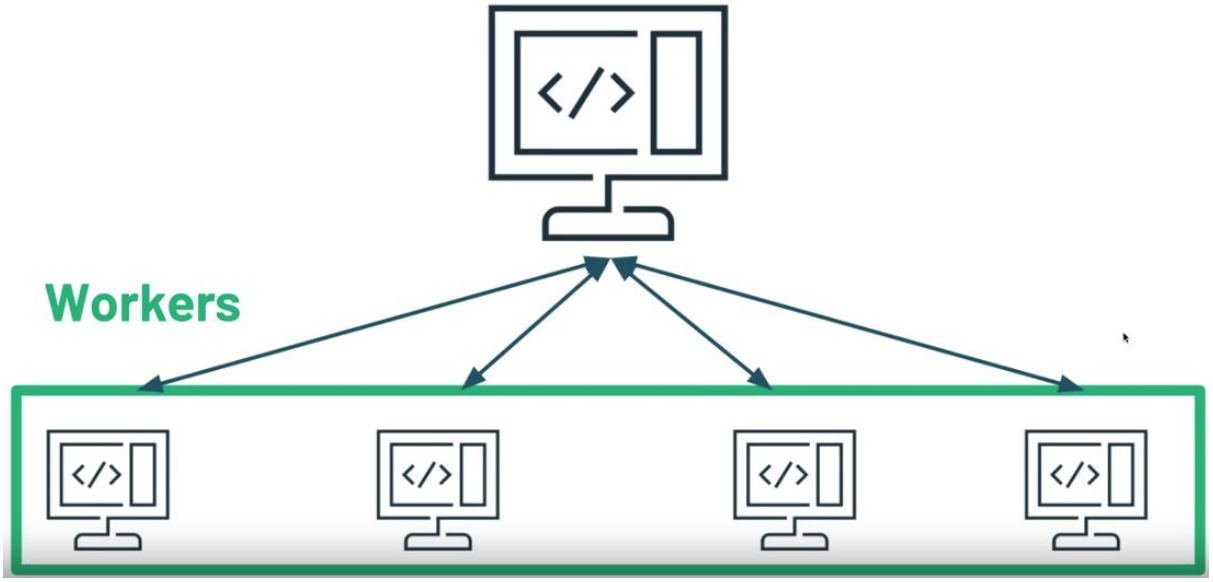
1. Spark SQL如何优化查询
    a. 一些术语
        i. Databricks table：一个结构化的数据表/集合
        ii. Schema：数据的结构
        iii. Metadata：关于table的信息
    b. 优化机制
        i. 变量核实
        ii. 创建一个逻辑计划（logic plan）



        iii. 优化流程



2. 集群如何通过工作流并行来扩展到大数据



3. 使用Spark UI
    a. 第一种：cluster-Spark UI
    b. 第二种：运行结果-View
4. 优化查询逻辑
   比如join前先filter
5. 决定什么时候使用缓存来加快查询
    a. 缓存：将一个table存储到集群的临时存储中
        i. 优点：速度快，适合频繁的读写
        ii. 缺点：占用时间和资源
        iii. 指令
            1. CACHE TABLE table;
            2. UNCACHE TABLE IF EXISTS table;
    b. Partitioning
        i. Partition的时候会对被partition列的每个值创建子文件夹。优点：只需要访问相关值对应的文件夹
        ii. 需要非常谨慎的选择用来排序的列

第5周 Complex Query

嵌套的数据结构

1. Nested JSON

name	loc	pets

"Kate"	{
	city : "New York",
	start_yr: "2005",
	boroughs: ["Brooklyn", "Bronx"]
	}
		["Charlie", "Loki"]

2. Data center
    a. 收集，存储，处理，分发数据
3. 探索一个嵌套的对象

```
SELECT EXPLODE (source)
FROM DCDataRaw;
```

4. CTE (Common Table Expressions)可以提供一个暂时的结果用于后续分析

```
WITH ExplodeSource  -- specify the name of the result set we will query
AS
(                   -- wrap a SELECT statement in parentheses
  SELECT            -- this is the temporary result set you will query
    dc_id,
    to_date(date) AS date,
    EXPLODE (source)
  FROM
    DCDataRaw
)
SELECT              -- write a select statment to query the result set
  key,
  dc_id,
  date,
  value.description,
  value.ip,
  value.temps,
  value.co2_level
FROM                -- this query is coming from the CTE we named
  ExplodeSource;
```

5. CTAS (Create Table as Select)

```
DROP TABLE IF EXISTS DeviceData;
CREATE TABLE DeviceData
USING parquet
WITH ExplodeSource                  -- The start of the CTE from the last cell
AS
```

```
(
SELECT
dc_id,
to_date(date) AS date,
EXPLODE (source)
FROM DCDataRaw
)
SELECT
dc_id,
key device_type,
date,
value.description,
value.ip,
value.temps,
value.co2_level

FROM ExplodeSource;
```

数据操作

1. 随机采样几行

```
SELECT * FROM outdoorProductsRaw TABLESAMPLE (5 ROWS)
```

2. 空值填充，日期分割

```
CREATE
OR REPLACE TEMPORARY VIEW outdoorProducts AS
SELECT
  InvoiceNo,
  StockCode,
  COALESCE(Description, "Misc") AS Description,
  Quantity,
  SPLIT(InvoiceDate, "/")[0] month,
  SPLIT(InvoiceDate, "/")[1] day,
  SPLIT(SPLIT(InvoiceDate, " ")[0], "/")[2] year,
  UnitPrice,
  Country
FROM
  outdoorProductsRaw
```

3. 左填充和concat

```
DROP TABLE IF EXISTS standardDate;
CREATE TABLE standardDate

WITH padStrings AS
(
SELECT
  InvoiceNo,
  StockCode,
  Description,
  Quantity,
  LPAD(month, 2, 0) AS month,
```

```
  LPAD(day, 2, 0) AS day,
  year,
  UnitPrice,
  Country
FROM outdoorProducts
)
SELECT
 InvoiceNo,
  StockCode,
  Description,
  Quantity,
  concat_ws("/", month, day, year) sDate,
  UnitPrice,
  Country
FROM padStrings;
```

4. 日期格式转换

```
CREATE
OR REPLACE TEMPORARY VIEW salesDateFormatted AS
SELECT
  InvoiceNo,
  StockCode,
  to_date(sDate, "MM/dd/yy") date,
  Quantity,
  CAST(UnitPrice AS DOUBLE)
FROM
  standardDate
```

5. 按天聚合

```
SELECT
  date_format(date, "E") day,
  SUM(quantity) totalQuantity
FROM
  salesDateFormatted
GROUP BY (day)
ORDER BY day
```

数据整理Data Munging

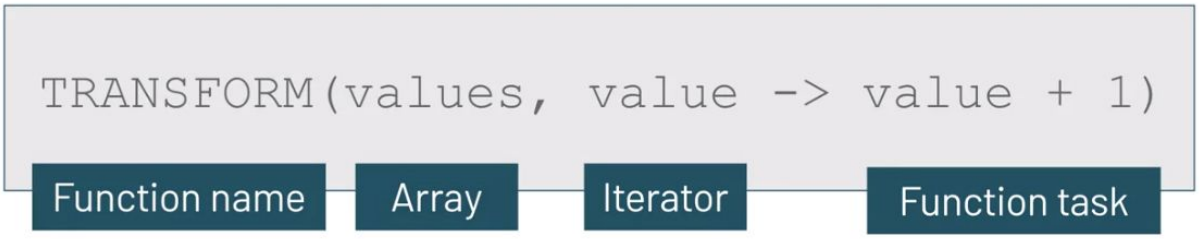
1. 创建表

```
DROP TABLE IF EXISTS outdoorProductsRaw;
CREATE TABLE outdoorProductsRaw USING csv OPTIONS (
  path "/mnt/training/online_retail/data-001/data.csv",
  header "true"
)
```

第6周 Spark SQL应用

高阶函数higher-order functions

1. 复杂数据类型：数组
    a. 常规操作：分解数组
    b. 缺点
        i. 容易出错
        ii. 容易丢失顺序信息
        iii. 低效率
2. Spark高阶函数



    a. 过滤Filter

```
SELECT
  categories,
  FILTER (categories, category -> category <> "Engineering Blog")
woEngineering
FROM DatabricksBlog
```

    b. 子语句subquery

```
SELECT
  *
FROM
  (
    SELECT
      authors, title,
      FILTER(categories, category -> category = "Engineering Blog") AS
blogType
    FROM
      DatabricksBlog
  )
WHERE
  size(blogType) > 0
```

    c. 存在Exists

```
SELECT
  categories,
  EXISTS (categories, c -> c = "Company Blog") companyFlag
FROM DatabricksBlog
```

    d. Transform

```
SELECT
  TRANSFORM(categories, cat -> LOWER(cat)) lwrCategories
FROM DatabricksBlog
```

    e. Reduce

```
CREATE OR REPLACE TEMPORARY VIEW Co2LevelsTemporary
AS
  SELECT
    dc_id,
    device_type,
    co2_Level,
    REDUCE(co2_Level, 0, (c, acc) -> c + acc, acc ->(acc div
size(co2_Level))) as averageCo2Level
  FROM DeviceData
  SORT BY averageCo2Level DESC
  ;

SELECT * FROM Co2LevelsTemporary
```

f. Pivot 1

```
SELECT * FROM (
  SELECT device_type, averageCo2Level
  FROM Co2LevelsTemporary
)
PIVOT (
  ROUND(AVG(averageCo2Level), 2) avg_co2
  FOR device_type IN ('sensor-ipad', 'sensor-inest',
    'sensor-istick', 'sensor-igauge')
  );
```

g. Pivot 2

```
SELECT
  *
FROM
  (
    SELECT
      month(date) month,
      REDUCE(co2_Level, 0, (c, acc) -> c + acc, acc ->(acc div
size(co2_Level))) averageCo2Level
    FROM
      DeviceData
  ) PIVOT (
    avg(averageCo2Level) avg FOR month IN (7 JUL, 8 AUG, 9 SEPT, 10
OCT, 11 NOV)
  )
```

h. Rollups：首先会对(A、B、C)进行GROUP BY，然后对(A、B)进行GROUP BY，然后是(A)进行GROUP BY，最后对全表进行GROUP BY操作

```
SELECT
  COALESCE(dc_id, "All data centers") AS dc_id,
  COALESCE(device_type, "All devices") AS device_type,
  ROUND(AVG(averageCo2Level))  AS avgCo2Level
FROM Co2LevelsTemporary
GROUP BY ROLLUP (dc_id, device_type)
ORDER BY dc_id, device_type;
```

i. Cube：首先会对(A、B、C)进行GROUP BY，然后依次是(A、B)，(A、C)，(A)，(B、C)，(B)，(C)，最后对全表进行GROUP BY操作

```
SELECT
  COALESCE(dc_id, "All data centers") AS dc_id,
  COALESCE(device_type, "All devices") AS device_type,
  ROUND(AVG(averageCo2Level))  AS avgCo2Level
FROM Co2LevelsTemporary
GROUP BY CUBE (dc_id, device_type)
ORDER BY dc_id, device_type;
```

排序表

1. 按列Partition

```
CREATE TABLE IF NOT EXISTS AvgTemps
PARTITIONED BY (device_type)
AS
  SELECT
    dc_id,
    date,
    temps,
    REDUCE(temps, 0, (t, acc) -> t + acc, acc ->(acc div size(temps))) as
avg_daily_temp_c,
    device_type
  FROM DeviceData;

SELECT * FROM AvgTemps;
```

2. 检查Partition

```
SHOW PARTITIONS AvgTemps
```

3. 创建widget

```
CREATE WIDGET DROPDOWN selectedDeviceType DEFAULT "sensor-inest"
CHOICES
SELECT
  DISTINCT device_type
FROM
  DeviceData
```

4. 使用选中的值

```
SELECT
  device_type,
  ROUND(AVG(avg_daily_temp_c),4) AS avgTemp,
  ROUND(STD(avg_daily_temp_c), 2) AS stdTemp
FROM AvgTemps
WHERE device_type = getArgument("selectedDeviceType")
GROUP BY device_type
```

5. 去除widget

```
REMOVE WIDGET selectedDeviceType
```

6. 窗函数

```
SELECT
  dc_id,
  month(date),
```

```
  avg_daily_temp_c,
  AVG(avg_daily_temp_c)
  OVER (PARTITION BY month(date), dc_id) AS avg_monthly_temp_c
FROM AvgTemps
WHERE month(date)="8" AND dc_id = "dc-102";
```

7. CTEs with window functions

```
WITH DiffChart AS
(
SELECT
  dc_id,
  date,
  avg_daily_temp_c,
  AVG(avg_daily_temp_c)
  OVER (PARTITION BY month(date), dc_id) AS avg_monthly_temp_c
FROM AvgTemps
)
SELECT
  dc_id,
  date,
  avg_daily_temp_c,
  avg_monthly_temp_c,
  avg_daily_temp_c - ROUND(avg_monthly_temp_c) AS degree_diff
FROM DiffChart;
```

第7周 数据存储和优化

现代数据存储

1. 数据仓库：将很多数据库结合起来，可以进行整体查询和查看
    a. 优点
        i. 标准SQL访问
        ii. 集合多数据源
        iii. 数据快速读取优化
        iv. 即时分析查询
    b. 缺点
        i. 不能存储初始数据
        ii. 难以扩展
        iii. 需要大量的投入
2. 数据湖
    a. 优点
        i. 可以存储所有数据：结构化，非结构化，半结构化
        ii. 为数据团队集中数据
        iii. 存储成本低，易扩展
    b. 挑战
        i. 数据可信度
        ii. 查询性能
3. 对比

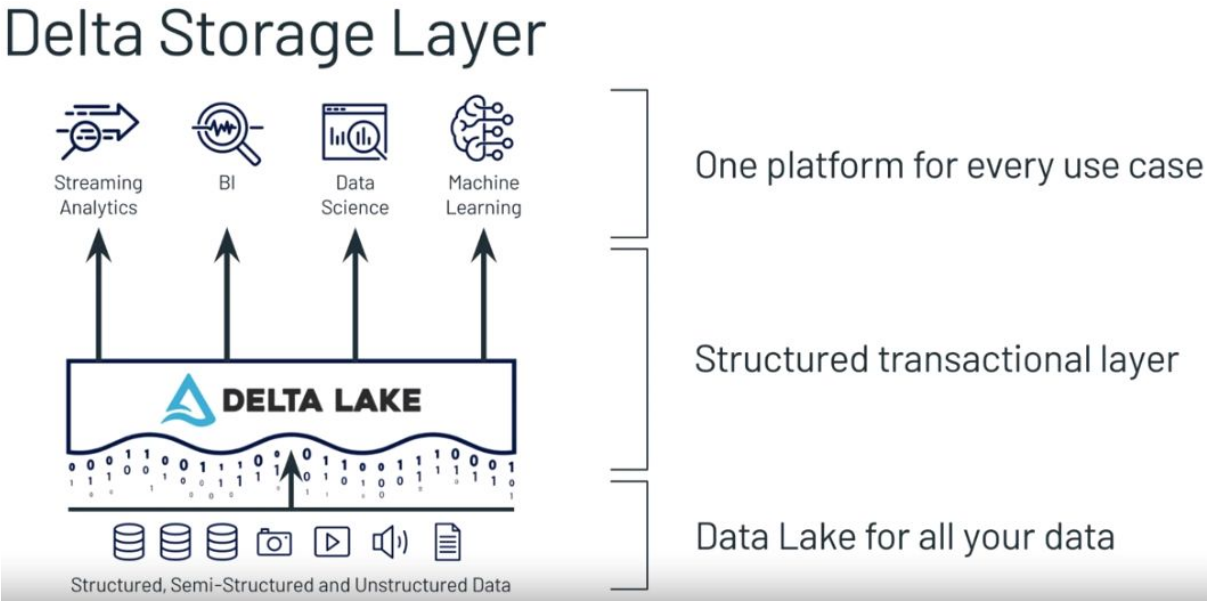
	Data lake	Data warehouse
Primary types of data	All types: Structured data, semi-structured data, unstructured (raw) data	Structured data only
Cost	$	$$$
Scalability	Scales to hold any amount of data at low cost, regardless of type	Scaling up becomes exponentially more expensive due to vendor costs
Intended users	Data analysts, data scientists	Data analysts
Vendor lock-in	No	Yes
Advantages	Low cost, flexibility, scalability, allows storage of the raw data needed for machine learning	User interface is familiar to users of traditional databases
Disadvantages	Exploring large amounts of raw data can be difficult without tools to organize and catalog the data	Expensive, always-on architecture, proprietary software, cannot hold unstructured (raw) data needed for machine learning

4. The Lakehouse：一种新的数据管理范式
    a. 使用了跟数据仓库相似的数据管理特性来保证快速查询，可靠性
    b. 建立在廉价灵活的存储上

使用Delta Lake

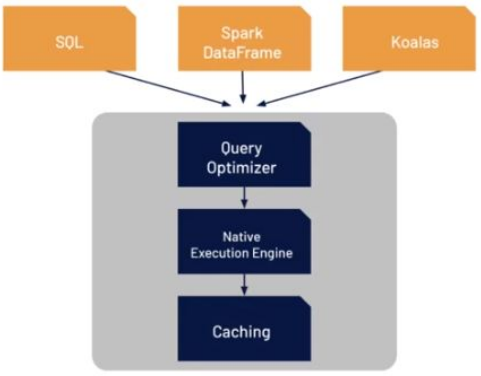
1. 什么是Delta Lake
    a. Data Lakehouse的关键组成
    b. 兼容ACID保证数据一致性
    c. 健壮数据存储
    d. 为Spark设计
2. 构成
    a. Delta架构

    b. Delta存储层



        i. 数据一致性保证
        ii. 元数据追踪
        iii. 自动处理表单变量
        iv. 允许版本控制和回滚
        v. 当数据进入时，合并和更新
    c. Delta引擎

- File management optimizations
- Performance optimization with Delta Caching
- Dynamic File Pruning
- Adaptive Query Execution



d. Delta表
   i. 创建Delta文件
   ii. 将表记录在元存储中
   iii. 启动一个交易日志

第8周 Delta Lake with Spark SQL

创建和维护Delta表

1. 创建表

```
DROP TABLE IF EXISTS health_tracker_data_2020_01;

CREATE TABLE health_tracker_data_2020_01
USING json
OPTIONS (
  path
"dbfs:/mnt/training/healthcare/tracker/raw.json/health_tracker_data_2020_1.json",
  inferSchema "true"
  );
```

2. 创建Delta表

```
CREATE OR REPLACE TABLE health_tracker_silver
USING DELTA
PARTITIONED BY (p_device_id)
LOCATION "/health_tracker/silver" AS (
SELECT
  value.name,
  value.heartrate,
  CAST(FROM_UNIXTIME(value.time) AS timestamp) AS time,
  CAST(FROM_UNIXTIME(value.time) AS DATE) AS dte,
  value.device_id p_device_id
FROM
  health_tracker_data_2020_01
)
```

3. 向表中插入

```
INSERT INTO
  health_tracker_silver
SELECT
  value.name,
  value.heartrate,
  CAST(FROM_UNIXTIME(value.time) AS timestamp) AS time,
  CAST(FROM_UNIXTIME(value.time) AS DATE) AS dte,
  value.device_id p_device_id
FROM
  health_tracker_data_2020_02
```

4. 查看特定版本的记录数

```
SELECT COUNT(*) FROM health_tracker_silver VERSION AS OF 0
```

5. 分组查看记录数

```
SELECT p_device_id, COUNT(*) FROM health_tracker_silver GROUP BY
p_device_id
```

6. 使用之前和之后的信息进行填充

```
CREATE OR REPLACE TEMPORARY VIEW updates
AS (
  SELECT name, (prev_amt+next_amt)/2 AS heartrate, time, dte, p_device_id
  FROM (
```

```
    SELECT *,
    LAG(heartrate) OVER (PARTITION BY p_device_id, dte ORDER BY p_device_id,
dte) AS prev_amt,
    LEAD(heartrate) OVER (PARTITION BY p_device_id, dte ORDER BY
p_device_id, dte) AS next_amt
    FROM health_tracker_silver
  )
  WHERE heartrate < 0
)
```

7. 预备upserts

```
CREATE OR REPLACE TEMPORARY VIEW upserts
AS (
    SELECT * FROM updates
    UNION ALL
    SELECT * FROM inserts
    )
```

8. 执行upserts

```
MERGE INTO health_tracker_silver                    -- the MERGE instruction is
used to perform the upsert
USING upserts

ON health_tracker_silver.time = upserts.time AND
   health_tracker_silver.p_device_id = upserts.p_device_id  -- ON is used to describe
the MERGE condition

WHEN MATCHED THEN                                   -- WHEN MATCHED describes
the update behavior
  UPDATE SET
  health_tracker_silver.heartrate = upserts.heartrate
WHEN NOT MATCHED THEN                                -- WHEN NOT MATCHED
describes the insert behavior
  INSERT (name, heartrate, time, dte, p_device_id)
  VALUES (name, heartrate, time, dte, p_device_id)
```

9. 历史描述

```
DESCRIBE HISTORY health_tracker_silver
```

10. 写入gold

```
DROP TABLE IF EXISTS health_tracker_gold;

CREATE TABLE health_tracker_gold
USING DELTA
LOCATION "/health_tracker/gold"
AS
SELECT
  AVG(heartrate) AS meanHeartrate,
  STD(heartrate) AS stdHeartrate,
  MAX(heartrate) AS maxHeartrate
FROM health_tracker_silver
GROUP BY p_device_id
```

Delta引擎优化

1. ZORDER
   OPTIMIZE flights ZORDER BY (DayofWeek);