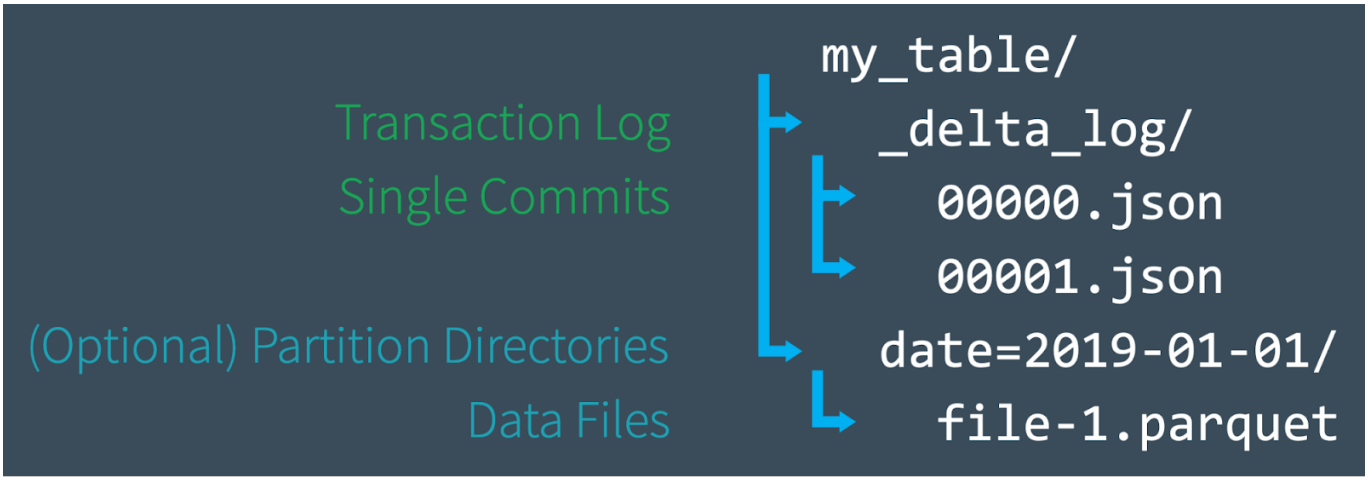
深入理解 Delta Lake 的 DML 实现原理 (Update, Delete, Merge)

Delta Lake 支持 DML 命令，包括 DELETE, UPDATE, 以及 MERGE，这些命令简化了 CDC、审计、治理以及 GDPR/CCPA
工作流等业务场景。在这篇文章中，我们将演示如何使用这些 DML
命令，并会介绍这些命令的后背实现，同时也会介绍对应命令的一些性能调优技巧。

Delta Lake: 基本原理



如果你是刚刚学习 Delta Lake，那么你可以看下本章以便快速了解 Delta Lake
的基本原理。本节主要在文件级别上介绍 Delta Lake 表的构建。

在创建新表时，Delta 将数据保存在一系列的 Parquet 文件中，并会在表的根目录创建 _delta_log 文件夹，其中包含 Delta Lake 的事务日志，ACID 事务日志里面记录了对应表的每次更改。当您修改表时（例如，通过添加新数据或执行更新、合并或删除），Delta Lake
将每个新事务的记录作为带编号的 JSON 文件保存在 delta_log 文件夹中，从 00...00000.json 命名开始，然后依次是 00...00001.json，00...00002.json，以此类推；每10个事务，Delta
还会在 delta_log 文件夹中生成一个“检查点” 的 Parquet
文件，这个文件允许我们快速重新创建表的状态。

最终，当我们查询 Delta Lake 表时，我们可以先读取事务日志，以快速确定哪些数据文件构成了表的最新版本，而不需要列出云对象存储中的所有文件，这大大提升了查询性能。当我们执行 DML 操作时，Delta Lake 会创建出新的文件，而不是在原来的文件里面修改它们，并使用事务日志来记录所有这些操作，比如哪些文件是新增的。如果想了解更多这方面的知识，可以参见《深入理解 Apache Spark Delta Lake 的事务日志》这篇文章。

好了，有了前面的基本介绍，下面我们就可以深入介绍 Delta Lake 的 DML

命令如何使用以及其背后的工作原理。下面的例子是使用 SQL 进行操作的，这个需要我们使用 Delta Lake 0.7.0 以及 Apache Spark 3.0，详情请参见 《在 Delta Lake 中启用 Spark SQL DDL 和 DML》。

UPDATE 使用及内部原理

可以使用 UPDATE
操作有选择地更新与筛选条件（也称为谓词，predicate
）匹配的任何行。下面的代码演示了如何将每种类型的谓词用作 UPDATE
语句的一部分。注意，Delta Lake 的 update 可以在 Python、Scala 和 SQL
中使用，但出于本文的目的，我们这里只是用 SQL 来介绍其使用。

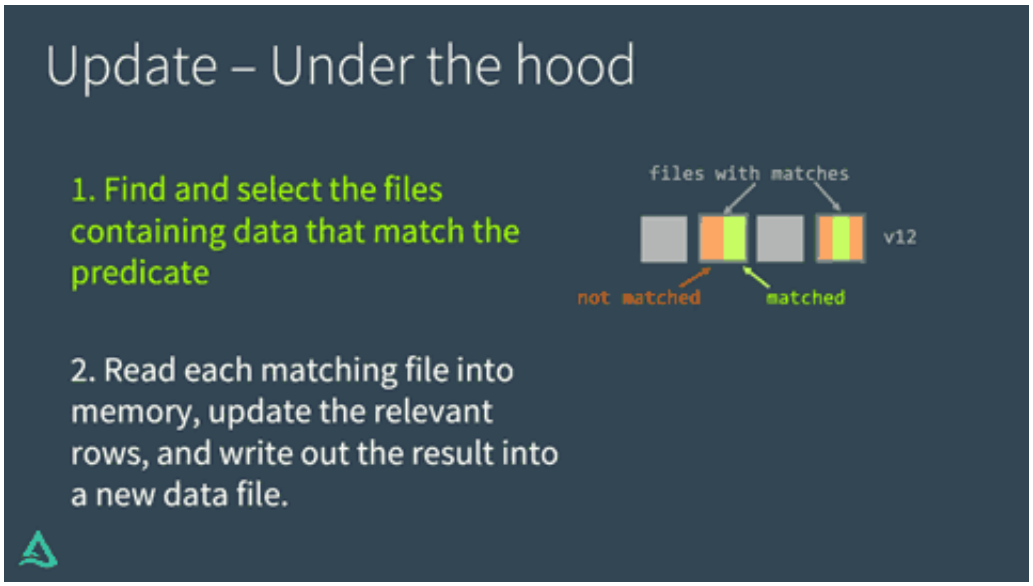
```
-- Update events
UPDATE events SET eventType = 'click' WHERE eventType = 'click'
```

UPDATE: Under the hood

UPDATE 在 Delta Lake 的实现是分为两步走的：

- 首先找到并选择那些包含与谓词匹配因而需要更新的数据的文件。这个过程中 Delta Lake 可以使用 data skipping 技术来加速这个过程。data skipping
  这个技术看起来是数砖的商业部才有，开源版本貌似没看到。
- 将每个匹配的文件读入内存，更新相关行，并将结果写入新的数据文件。

整个过程如下：



一旦 Delta Lake 成功地执行了 UPDATE，它就会在事务日志中添加一个提交，表明从现在开始将使用新的数据文件来代替旧的数据文件。但是，旧的数据文件没有被删除。相反，它只是被标记成 tombstoned ——
意思是这个文件只属于表的旧版本数据文件而不是当前版本的数据文件。Delta Lake 能够使用它来提供数据版本控制和时间旅行（time travel）。

UPDATE + Delta Lake 时间旅行 = Easy debugging

保留旧的数据文件对于调试非常有用，因为您可以在任何时候使用 Delta Lake 的时间旅行回到历史并查询表的以前版本数据。如果哪天我们不小心错误地更新了表，并希望找出发生了什么，我们可以轻松地比较表的两个版本。

```
SELECT * FROM events VERSION AS OF 12
```

UPDATE：性能调优

提高 Delta Lake 的 UPDATE 命令性能的主要方法是添加更多的谓词来缩小搜索空间。搜索越具体，Delta Lake 需要扫描和/或修改的文件就越少。

Databricks 的商业版 Delta Lake 具有一些企业性增强，如改进的 data skipping、布隆过滤器（bloom filters）的使用和 Z-Order Optimize ，Z-Order Optimize 重新组织每个数据文件的布局，使相似的列值在策略上相互接近，以获得最大效率。

DELETE 使用及内部原理

我们可以使用 DELETE 命令并根据谓词（过滤条件）选择性地删除任意行。

```
DELETE FROM events WHERE date < '2017-01-01'
```

如果希望恢复意外的删除操作，可以使用时间旅行将表回滚到原来的状态，如下面的 Python 代码片段所示。

```
# Read correct version of table into memory
dt = spark.read.format("delta") ₩
        .option("versionAsOf", 4) ₩
        .load("/tmp/loans_delta")

# Overwrite current table with DataFrame in memory
```

```
dt.write.format("delta") ₩
      .mode("overwrite") ₩
      .save(deltaPath)
```

DELETE: Under the hood

DELETE 的工作原理和 UPDATE 一样。Delta Lake 对数据进行两次扫描：第一次扫描是识别包含与谓词条件匹配的行的任何数据文件。第二次扫描将匹配的数据文件读入内存，此时 Delta Lake 删除相关的行，然后将未删除的数据写入磁盘上的新文件中。

在 Delta Lake
成功完成删除操作后，旧的数据文件不会被删除——它们仍然保留在磁盘上，但是在 Delta Lake 事务日志中这些文件被记录为 tombstoned (不再是活动表的一部分)。记住，那些旧文件不会立即删除，因为我们可能需要使用时间旅行功能来跳到更早版本的数据。如果想删除超过一定时间期限的文件，可以使用 VACUUM 命令。

DELETE + VACUUM：清理旧的文件

运行 VACUUM 命令永久删除满足以下条件的所有数据文件：

- 不再是活动表的一部分，并且
- 超过保留阈值（默认为七天）。

Delta Lake 不会自动删除旧文件——我们必须自己运行 VACUUM 命令，如下所示。如果我们希望指定一个与默认值不同的保留期限，那么我们可以将其作为参数提供。

```
from delta.tables import *

# vacuum files not required by versions older than the default
# retention period, which is 168 hours (7 days) by default
dt.vacuum()
deltaTable.vacuum(48) # vacuum files older than 48 hours
```

注意：运行 VACUUM 命令时指定的保留时间为0小时将删除表的最新版本中未使用的所有文件。请确保运行这个命令时，对应的表没有写操作，否则可能会发生数据丢失。

DELETE：性能调优

与 UPDATE 命令一样，提高 Delta Lake DELETE
操作性能的主要方法是添加更多谓词来缩小搜索空间。Databricks 的商业版 Delta Lake

具有一些企业性增强，如改进的 data skipping、布隆过滤器（bloom filters）的使用和 Z-Order Optimize 。

MERGE 使用及内部原理

Delta Lake 的 MERGE 命令允许我们实现 upserts 语义，其实 UPDATE 和 INSERT 的混合。为了理解 upserts 的含义，假设我们有一张表（目标表）和一张源表，其中包含新记录和对现有记录的更新。upsert 是这样工作的：

- 当源表中的一条记录与目标表中现有的记录相匹配时，Delta Lake 将更新该记录。
- 当没有匹配时，Delta Lake 将插入新记录。

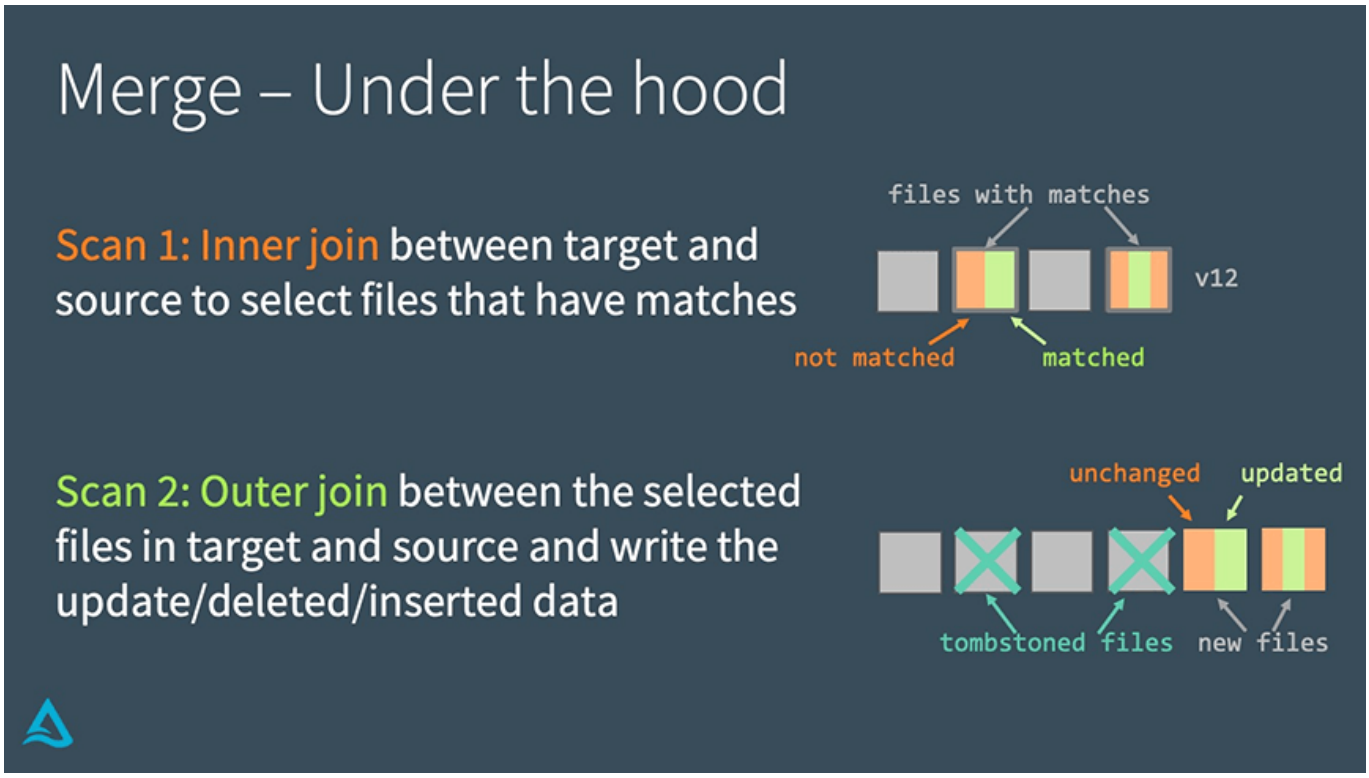
```
MERGE INTO events
USING updates
  ON events.eventId = updates.eventId
  WHEN MATCHED THEN UPDATE
    SET events.data = updates.data
  WHEN NOT MATCHED THEN
    INSERT (date, eventId, data) VALUES (date, eventId, data)
```

Delta Lake 的 MERGE 命令极大地简化了工作流。

MERGE: Under the hood

Delta Lake 通过下面两步实现 MERGE：

- 在目标表和源表之间执行 inner join，以选择所有匹配的文件。
- 在目标表中选中的文件和源表之间执行 outer join ，并写出更新/删除/插入的数据。



如果想及时了解Spark、Hadoop或者HBase相关的文章，欢迎关注微信公众号：iteblog_hadoop

MERGE 的实现与 UPDATE 或者 DELETE 的主要区别是其使用了 join，这一事实允许我们在寻求提高性能时使用一些独特的策略。

MERGE：性能调优

为了提升 MERGE 的执行性能，我们需要了解上面两个 join 中的哪个影响了程序的执行。

如果 inner join 是执行 MERGE 的瓶颈（比如找到 Delta Lake 需要重写的文件花费的时间太长了），那么我们可以采用以下的策略解决：

- 添加更多的过滤条件来减少搜索空间；
- 调整 shuffle partition 的数量；
- 调整 broadcast join 的阈值；.
- 如果表中有很多小文件，我们可以先压缩合并它们；但不要将它们压缩为太大的文件，因为 Delta Lake 必须复制整个文件来重写它。

如果 outer join 是执行 MERGE 的瓶颈（比如重写文件花费的时间太长了），那么我们可以采用以下的策略解决：

- 调整 shuffle partition 的数量
    - 这个对分区表可能会生成很多的小文件；
    - 在写文件之前开启自动重分区来减少 Reduce 产生的文件。
- 调整 broadcast 的阈值。如果我们使用 full outer join, Spark 则无法进行 broadcast join,

但是如果我们使用 right outer join, Spark 则可以使用；我们可以根据实际情况来调整 broadcast 的阈值；
- 缓存 source table / DataFrame.
    - 缓存 source table 可以加快第二次的扫描时间，但是记住别缓存 target table，因为这可能会导致缓存一致性问题。

总结

Delta Lake支持 UPDATE、DELETE 以及 MERGE INTO 等 DML 命令，这极大地简化了许多常见大数据操作的工作流程。在本文中，我们演示了如何在 Delta Lake 中使用这些命令，介绍了这些 DML 命令的实现原理，并提供了一些性能调优技巧。

本文翻译自：Diving Into Delta Lake: DML Internals (Update, Delete, Merge)