Delta Lake

"The end of 'Fashion-IT' - customers will only pay for value and not technology." Sunny Gosh

## Um pouco de história

Primeiramente surgiram os Data Warehouses: bases de dados para sistemas OLAPs que desenhados com modelagem Star Schema ou Snowflake visavam organizar dados por assunto e propiciar melhores consultas se comparados a transacionais (OLTPs). Os DWs costumavam ser construídos em bancos de dados relacionais, como Oracle e Teradata, e por isso se limitavam a dados estruturados.

Com o crescente aumento do volume de dados gerados e manipulados pelas empresas, além do valor a ser extraído de dados semi e não estruturados, bancos de dados - e consequentemente os DWs portanto - passaram a não ser mais suficientes. Assim nascem os Data Lakes: repositórios que, organizados em camadas, são capazes de armazenar todo e qualquer tipo de dado. Todavia, a necessidade de organizar dados de diversas fontes por assunto, e conectar a ferramentas de visualização ainda se fazia necessária, tornando os DWs ainda relevantes. Chegamos então a uma arquitetura como essa: dados de diversas fontes são coletados e armazenados na primeira camada de um Data Lake através de uma ferramenta de ingestão, conforme os dados avançavam pelas camadas são tratados e transformados até, por fim, serem inseridos em uma ferramenta de DW distribuído (que resolve as limitações de um SGBD tradicional com relação a volumetria), como Redshift da AWS ou Synapse Analytics (antigo Azure DW) da Microsoft, para então alimentarem dashboards.

Arquiteturas similares à descrita são funcionais e encontradas em diversos ambientes modernos. Entretanto, ela apresenta alguns problemas:

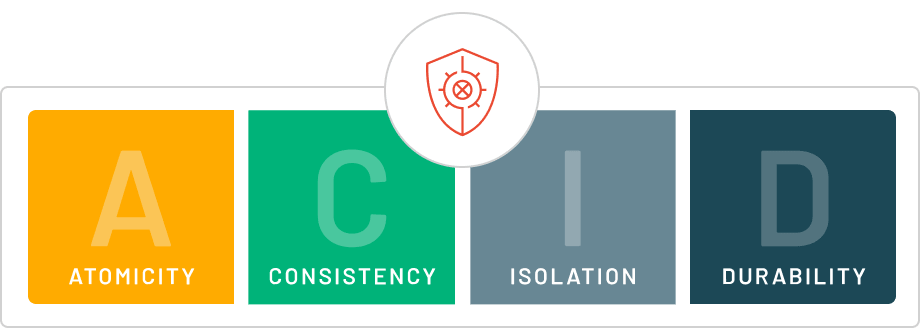
* Com a necessidade de mover dados de um Data Lake para um Data Warehouse, custos são impostos sobre o uso de ferramentas de ingestão para realizar tais cargas.
* Estando os mesmos dados presentes no Data Lake e em um Data Warehouse, há duplicidade que implica em custos extras de armazenamento.

Existe alguma arquitetura que resolva esses pontos?

Tomando como base os assuntos discutidos no curso até aqui a primeira resposta que talvez lhe venha em mente seja: Apache Hive, ferramenta do ecossistema Hadoop que permite a execução de queries sobre arquivos, além da gestão de metadados para tratá-los como tabelas e bancos de dados. E de fato o Hive é uma solução muito interessante, porquanto não se limita ao ecossistema Hadoop (seja via Databricks ou não) e poderia permitir a criação de um DW sobre uma camada do Data Lake, para isso foi criado. Mas o Hive possui aspectos negativos, como não promover transações ACID.

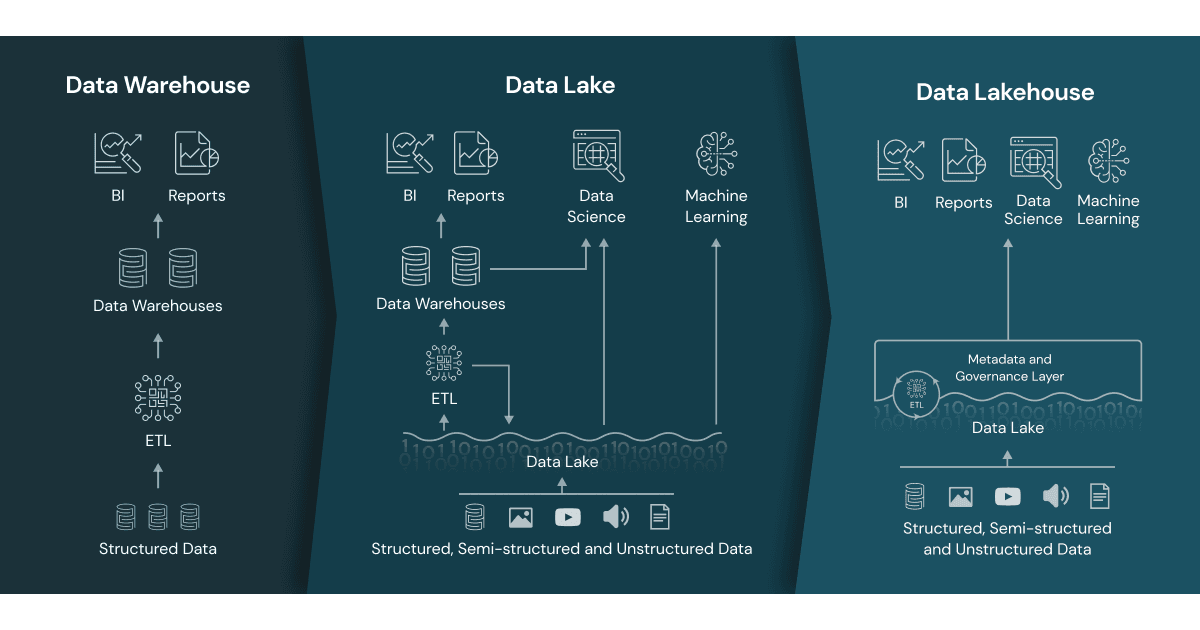
### ACID

ACID são propriedades de transação de um banco de dados que visam garantir confiabilidade, visto que com múltiplos usuários simultâneos é muito comum encontrarmos concorrência de acesso a objetos e tabelas.

* Atomicidade: em uma transação composta por duas ou mais partes, ou a transação é executada por completo ou não será.
* Consistência: se a transação não criar um estado válido dos dados ao seu final, seu estado anterior deverá ser restabelecido.
* Isolamento: uma transação ainda em andamento deve se manter isolada de outras operações, evitando interferências por concorrência.
* Durabilidade: dados validados são registrados de forma que mesmo em caso de falha futura ou reinício do sistema eles se manterão corretos.  Fonte: [Databricks](https://databricks.com/glossary/acid-transactions)

## Data Lakehouse

Unindo o melhor de ambos os mundos: a flexibilidade e baixo custo de armazenamento de um Data Lake e organização de um Data Warehouse surgem os Data Lakehouses. Capazes então de consultar dados fisicamente em um DL, como o Apache Hive, mas trazendo mais confiabilidade através de transações ACID, uma arquitetura como esta pode ser implementada utilizando Redshift Spectrum da AWS, Snowflake ou Delta Lake (solução open source) no Databricks, por exemplo.

 Fonte: [Databricks](https://databricks.com/glossary/data-lakehouse)

### Delta Lake

Um Data Lakehouse pode ser implementado através de Delta Lake em qualquer plataforma, mas pela excelente sinergia presente com Databricks, esta combinação é bastante comum. Existem inúmeras formas para criar tabelas do tipo Delta, na primeira delas basta utilizar a cláusula USING já vista, seguida de DELTA:

CREATE TABLE <nome\_tabela> (<campos e tipos de dados>)

USING DELTA;

Ou simplesmente não indicar o formato da tabela, pois o padrão é Delta:

CREATE TABLE <nome\_tabela> (<campos e tipos de dados>);

A tabela criada estará disponível no menu Data a direita, bem como o banco de dados onde foi inserida, e terá seus arquivos armazenados em parquet com compressão snappy.

Também é possível criar e manipular tabelas Delta a partir de seu caminho, como no exemplo:

CREATE TABLE delta.`path/to/table` USING DELTA;

Ou mesmo passar comandos para a função spark.sql:

spark.sql("""

CREATE TABLE <nome\_tabela> (<campos e tipos de dados>) USING delta

""")

Caso esteja fora da plataforma Databricks e queira manipular tabelas do tipo Delta com PySpark, instale uma versão de PySpark compatível com Delta:

pip install pyspark==<compatible-spark-version>

E depois execute o PySpark com o pacote Delta Lake e algumas configurações adicionais:

pyspark --packages io.delta:delta-core\_2.12:2.2.0 --conf "spark.sql.extensions=io.delta.sql.DeltaSparkSessionExtension" --conf "spark.sql.catalog.spark\_catalog=org.apache.spark.sql.delta.catalog.DeltaCatalog"

Caso precise de mais ajuda, não hesite consultar a seção de indicações e bibliografia ao final deste material.

Também é possível criar e carregar tabelas a partir de Dataframes, como abaixo:

df.write.format("delta").save("path")

Lembrando que mais opções podem ser informadas como o modo do salvamento (mode) e ou particionamento (partitionBy).

Ou mesmo converter tabelas já existentes em:

CONVERT TO DELTA database\_name.table\_name;

#### Funcionamento

Como pode perceber, a criação de tabelas Delta, bem como sua manipulação, se dá de forma muito similar às tabelas convencionais baseadas em arquivos que vimos anteriormente. Entretanto, se olhar atentamente verá que no diretório onde a tabela é criada um diretório de log de transações também é criado, pois é desta forma que ACID é garantido em DataLakehouses utilizando Delta Lake. Além disso, esta funcionalidade permite time travel(viagem no tempo), o que possibilita consultar versões anteriores de dados, e mais.

#### Vantagens

Além de todas as vantagens já descritas de um Data Lakehouse, há mais um aspecto interessante das Delta Tables no Databricks: enquanto tabelas particionadas que usam hive precisam ter seus metadados atualizados todas vezes que recebem novos dados (MSCK REPAIR TABLE), tabelas criadas usando Delta não!

## Indicações e Bibliografia

[Mentalidade Data-driven](https://letscode.com.br/blog/mentalidade-data-driven-entenda-a-importancia-para-sua-empresa)

[Transações ACID](https://databricks.com/glossary/acid-transactions)

[Amazon Redshift Spectrum](https://docs.aws.amazon.com/pt_br/redshift/latest/dg/c-using-spectrum.html)

[Data Lakehouse - Snowflake](https://www.snowflake.com/guides/what-data-lakehouse)

[Data Lakehouse - Databricks](https://databricks.com/glossary/data-lakehouse)

[Data Lakehouse - GCP](https://cloud.google.com/blog/products/data-analytics/open-data-lakehouse-on-google-cloud)

[Getting Started Delta Lake](https://delta.io/learn/getting-started/)

[Criar tabelas delta](https://delta.io/blog/2022-10-25-create-delta-lake-tables/)