Resumo Completo sobre os Temas Essenciais para um Engenheiro de Dados

1. Arquitetura de Dados

- **Data Pipeline**: Refere-se à sequência de processos que movem os dados de uma origem até um destino. Um pipeline geralmente envolve *ingestão*, *transformação* e *armazenamento* de dados.
- **ETL vs. ELT**:
    - **ETL (Extract, Transform, Load)**: Processo de *extração* de dados, seguido de sua *transformação* (limpeza, validação, agregação) e, por fim, *carregamento* no destino (geralmente um Data Warehouse). Comum em ambientes tradicionais de Data Warehousing.
    - **ELT (Extract, Load, Transform)**: Dados são extraídos e carregados diretamente para o destino (normalmente um Data Lake) antes de serem transformados, o que é eficiente quando o destino tem poder de processamento para fazer a transformação depois do carregamento.
- **Data Lake vs. Data Warehouse**:
    - **Data Lake**: Armazena dados em seu estado bruto, sem estrutura pré-definida (dados estruturados, semi-estruturados e não estruturados). Ideal para dados de grande volume, diversidade e para análise futura.
    - **Data Warehouse**: Armazena dados já processados, transformados e organizados para análise específica, normalmente em formato relacional e otimizado para consultas analíticas.

2. Processamento de Dados

- **Apache Spark**:
    - **DataFrame API**: Permite a manipulação de dados tabulares de forma distribuída. Similar ao Pandas, mas projetado para escalabilidade.
    - **RDDs (Resilient Distributed Datasets)**: A base do processamento do Spark, que permite distribuir o processamento dos dados de forma eficiente e resiliente.
    - **Lazy Evaluation**: Transformações em Spark são executadas de maneira adiada até que a ação final (como `.show()` ou `.collect()`) seja chamada. Isso ajuda a otimizar o processamento.
- **Parquet e JSON**:
    - **Parquet**: Formato de arquivo colunar altamente eficiente, otimizado para leitura e escrita em ambientes de Big Data. Ideal para consultas analíticas.
    - **JSON**: Formato flexível baseado em chave-valor. Embora seja muito usado para dados semi-estruturados, não é tão eficiente quanto Parquet para consultas de grandes volumes de dados.
- **Particionamento e Otimização**:
    - **Particionamento**: Dividir dados em segmentos (partições) para melhorar a performance de leitura e escrita, especialmente em ambientes distribuídos.

    - **Cache/Persistência no Spark**: Utilizado para evitar a reexecução de operações dispendiosas, como carregamento ou processamento de dados já lidos, ao manter os dados na memória.

3. Cloud (Azure)

- **Azure Data Factory (ADF)**: Plataforma de orquestração que facilita a movimentação e transformação de dados entre diferentes fontes e destinos em ambientes de Big Data.
- **Azure Synapse Analytics**: Solução de Data Warehouse escalável na nuvem, integrando grandes volumes de dados para análise, com capacidades de SQL e Spark.
- **Azure Blob Storage**: Armazenamento de objetos na nuvem, utilizado para armazenar dados em seu estado bruto ou já transformado, sendo uma solução de baixo custo e altamente escalável.

4. Bancos de Dados

- **SQL**:
    - **Joins**: Operações fundamentais para combinar dados de múltiplas tabelas. Tipos incluem `INNER JOIN`, `LEFT JOIN`, `RIGHT JOIN` e `FULL JOIN`.
    - **Window Functions**: Funções que realizam cálculos sobre um conjunto de linhas relacionadas, como `ROW_NUMBER()`, `RANK()`, `LAG()`, `LEAD()`. Útil para cálculos que precisam de um contexto de janela.
    - **Indexação**: Técnicas que melhoram a performance das consultas em tabelas grandes, criando estruturas de dados auxiliares para acelerar o acesso aos dados.
- **NoSQL**:
    - **MongoDB/CosmosDB**: Bancos de dados orientados a documentos, que armazenam dados em formato de JSON ou BSON. Ideal para dados semi-estruturados e com esquema flexível.

5. Versionamento e Monitoramento

- **Git**:
    - **Branching**: Criação de ramificações no repositório para desenvolvimento paralelo.
    - **Merge**: Processo de combinar alterações feitas em diferentes branches.
    - **Pull Request**: Solicitação para integrar alterações em um branch principal, frequentemente usada para revisão de código antes de ser aceita.
- **Monitoramento de Pipelines**:
    - **Logs e alertas**: Ferramentas para monitorar a execução de pipelines de dados, como no **Apache Airflow** ou **Azure Data Factory**, com notificações de falhas ou erros.
    - **Retries**: Capacidade de tentar novamente uma execução de pipeline caso ela falhe, garantindo maior resiliência no processo de movimentação de dados.

Resumo sobre Tipos de Dados e Seus Bancos

1. Tipos de Dados

- **Estruturados**: Dados organizados em tabelas com um esquema definido (ex: bancos relacionais). São fáceis de armazenar, consultar e analisar, e geralmente possuem formatos como inteiros, strings, datas, etc.
- **Semi-estruturados**: Dados com estrutura flexível, como JSON, XML, e YAML. Não seguem um esquema rígido, mas têm tags ou identificadores que permitem organizar e interpretar os dados.
- **Não estruturados**: Dados que não têm formato ou estrutura predefinidos. Exemplos incluem textos livres, imagens, vídeos, e-mails e logs. Requerem processamento adicional para extração de informações.

2. Tipos de Bancos de Dados e Seus Usos

- **Bancos de Dados Relacionais (SQL)**:
    - **Exemplos**: MySQL, PostgreSQL, SQL Server, Oracle.
    - **Características**: Organizam dados em tabelas com linhas e colunas. Usam SQL para consultas. Ideais para dados estruturados com relações complexas e transações consistentes.
- **Bancos de Dados NoSQL**:
    - **Documentos (ex: MongoDB, Cosmos DB)**: Armazenam dados em formato JSON/BSON, sendo flexíveis para armazenar dados semi-estruturados, como logs e configurações.
    - **Chave-valor (ex: Redis, DynamoDB)**: Armazenam pares chave-valor, adequados para dados simples e de acesso rápido, como caches e sessões de usuário.
    - **Colunares (ex: Cassandra, HBase)**: Armazenam dados em colunas, otimizados para consultas rápidas de grandes volumes de dados, especialmente em ambientes de Big Data.
    - **Grafos (ex: Neo4j, Amazon Neptune)**: Armazenam dados em grafos, onde as entidades são nós e as conexões entre elas são arestas. Ideais para redes sociais e análise de relações complexas.

3. Considerações

- **SQL**: Ideal para dados com estrutura rígida e operações transacionais.
- **NoSQL**: Melhor para dados em grande volume, flexíveis ou de alto desempenho, onde a estrutura pode variar e a escalabilidade é necessária.