Data Engineer - Test Task

Q.1 **What are the most significant experiences you had in building data platforms? What roles have you played during this time?**

Ans. **Building a Centralized Data Hub on AWS**

In my previous role, I helped design and build a central storage system for all our company's data on Amazon Web Services (AWS). This "data lake" would hold everything from customer transactions to website logs, making it easier for everyone to analyze information.

My Role as a Data Engineer

Here's what I did as a data engineer:

- **Data Storage**: I set up secure and flexible storage on AWS S3 to hold all the data**.**
- **Data Pipelines:** I built automated systems to move data from different sources into the data lake smoothly**.**
- **Saving Money:** I found ways to store data efficiently on AWS, saving the company money.
- **Cleaning Up Data:** I made sure the data was accurate and organized before putting it in the lake**.**
- **Data Catalog:** I set up an automatic system to organize and label the data, making it easier to find what you need.
- **Data Structure:** I defined a clear structure for the data to keep it consistent and reliable.
- **Fast Searching:** I chose the best format to store the data so it could be searched quickly.
- **Efficient Storage:** I organized the data in the lake in a way that made it easy to access specific parts**.**
- **Automating Everything:** I used tools to automatically set up and manage the data lake, saving time and effort.
- **Keeping an Eye on Things:** I set up monitoring to make sure the data lake was running smoothly and to catch any problems.
- **Redshift Connection:** I made it possible to access the data lake directly from our data analysis tool (Redshift & QuickSight).

- **Customizable Views:** I created different ways to view the data depending on what each team in the company needed.

Q.2 How does the ETL processes differ when ingesting data from a NoSQL database & a SQL database? Do you have any experience around that, and if yes, please explain briefly based on your experience the differences and the difficulties of each type of technology.

Ans. Differences in ETL Processes When Ingesting Data from NoSQL and SQL Databases

Schema Differences
- SQL Databases:
- Structured Schema: SQL databases have a well-defined schema, making it easier to understand and transform data during ETL.
- Experience: I migrated data from MySQL to a cloud data warehouse, and the structured nature made mapping and transforming data straightforward**.**

- NoSQL Databases:
- Flexible Schema: NoSQL databases, like MongoDB, have flexible schemas, leading to challenges in data transformation.
- Experience: I was part of integrating MongoDB data into a data lake but didn't get a chance to work with custom transformation logic.

Data Consistency and Integrity
- SQL Databases:
- Strong Consistency: SQL databases ensure strong consistency and integrity constraints, simplifying ETL.

- **Experience:** While extracting data from an Mysql database, primary and foreign keys ensure data integrity, making it easier to maintain relationships.

- NoSQL Databases:
- **Eventual Consistency:** NoSQL databases prioritize availability, leading to eventual consistency and complicating the ETL process.
- **Experience:** I worked with AWS DynamoDb and had to reconcile data versions due to eventual consistency.

Query and Access Patterns
- SQL Databases:
- **Standardized Queries:** SQL databases use standardized SQL queries, well-supported by ETL tools.
- **Experience:** I used tools like DBeaver to query and extract data from SQL databases efficiently.

- NoSQL Databases:
- **Varied Access Patterns:** NoSQL databases use non-SQL query languages or APIs, requiring custom connectors or scripts.
- **Experience:** In a MongoDB project, I developed custom Python scripts with PyMongo to extract and transform data.

Performance Considerations
- SQL Databases:
- **Optimized for Transactions:** SQL databases are efficient for read-heavy ETL operations.
- **Experience:** Extracting data from PostgreSQL was efficient, allowing complex transformations without significant performance overhead.

- NoSQL Databases:
- **Optimized for Scalability:** NoSQL databases require additional tuning for efficient data extraction**.**
- **Experience:** not worked as such

Personal Experience with ETL from NoSQL and SQL Databases

SQL Database ETL Experience:
- **Project:** Migrating data from MySQL to Amazon Redshift.
- **Role:** Data Engineer.
- **Process:** I used AWS DMS to extract data and AWS Glue for transformation before loading it into Redshift.
- **Challenges:** Managing large data volumes and ensuring minimal downtime during migration.

NoSQL Database ETL Experience:
- **Project:** Integrating MongoDB data into a data lake on AWS S3.
- **Role:** Data Engineer.
- **Process:** I developed custom Python scripts to extract, transform, and load data into S3, using AWS Glue for further transformations.
- **Challenges:** Handling varying document structures, ensuring data consistency, and managing performance impacts on MongoDB during extraction.

Ingesting data from SQL databases is simpler due to structured schemas and strong consistency. NoSQL databases add complexity with flexible schemas and eventual consistency. My experience with both has equipped me to handle these challenges effectively.

Q3. What techniques would you use for efficiently processing large chunks of data? What are the technologies of your choice when it comes to this, and explain briefly the reasoning behind choosing them, while also giving some examples of usage from your own experience.

Ans. Techniques for Efficiently Processing Large Chunks of Data

1. Parallel Processing:
- **Technique:** Divide the data into smaller chunks and process them simultaneously.
- **Technologies:** Apache Spark, Apache Hadoop,multiprocessing in python.
- **Reasoning:** Parallel processing improves speed and efficiency by leveraging multiple CPUs/cores.

- **Example:** I used AWS Glue Jobs based on Apache Spark to process terabytes of log data in parallel, reducing the processing time from hours to minutes.

2. Batch Processing:
- **Technique:** Process data in large batches rather than in real-time.
- **Technologies:**AWS Glue Jobs based on Apache Spark**,** AWS EMR**.**
- **Reasoning:** Batch processing is efficient for handling large volumes of data at scheduled intervals**.**
- **Example: I**n a data warehousing project, I used AWS EMR to perform nightly batch processing of transactional data, optimizing resource usage and ensuring data consistency.

3. Distributed Computing:
- **Technique:** Distribute data and computation across a cluster of machines**.**
- **Technologies:** Apache Kafka, Amazon Redshift, Google BigQuery.
- **Reasoning:** Distributed systems can scale horizontally to handle larger datasets.
- **Example:** For a real-time analytics platform, I used AWS Redshift to stream data and distribute processing across multiple nodes, ensuring high availability and fault tolerance.

4. In-Memory Processing:
- **Technique:** Process data in memory to reduce I/O overhead**.**
- **Technologies:** Apache Spark, AWS QuickSight.
- **Reasoning:** In-memory processing provides significant speed improvements for iterative and interactive computations.
- **Example:** Not much but AWS glue jobs are built on top apache spark I used those jobs**.**

5. Columnar Storage Formats:
- **Technique:** Store data in a columnar format to optimize for read-heavy operation**s.**
- **Technologies:** Apache Parquet, Apache ORC**.**
- **Reasoning:** Columnar storage reduces I/O and improves query performance by reading only the required columns.
- **Example:** I stored data in Apache Parquet format for an analytics project, which improved query performance in Apache Spark, reducing response times by 50%**.**

6. Data Partitioning:

- **Technique:** Partition data to enable parallel processing and improve query performance**.**
- **Technologies:** Hive, Apache Spark, Amazon S3.
- **Reasoning:** Partitioning helps in distributing the data across nodes and optimizing read/write operations.
- **Example:** In a data lake project, I partitioned data in Amazon S3 by date, improving query performance and reducing costs by scanning smaller data segments.

7. Indexing and Caching:

- **Technique:** Use indexing and caching to speed up data retrieval.
- **Technologies: Elasticsearch, Redis,Redshift Cluster.**
- **Reasoning:** Indexing reduces the time to locate data, while caching keeps frequently accessed data in memory.
- **Example:** Not much but i used indexing in few of redshift tables

Personal Experience

1. AWS Glue for Parallel Processing:

- **Project:** Log Data Analysis.
- **Role:** Data Engineer.
- **Process:** Used Spark to parallelize the processing of log files stored in s3**.**
- **Outcome:** Reduced processing time from several hours to minutes.

2. AWS EMR for Batch Processing:

- **Project:** Data Warehousing.
- **Role:** Data Engineer.
- **Process:** Implemented nightly batch ETL processes using AWS EMR.
- **Outcome:** Ensured timely and efficient data processing, optimizing resource usage.

3. Columnar Storage with Parquet:
- **Project:** Analytics Platform.
- **Role:** Data Engineer.
- **Process:** Stored data in Parquet format for efficient querying.
- **Outcome:** Improved query performance, reducing execution times by 50%.

Q4. What is data integrity to you, and how do you maintain & measure data integrity in such ecosystem that is ever changing, and what are the most important metrics that come to mind that should be sought after? Please provide examples of ways that you have measured, the methods you have used to maintain those, and any techniques you have employed to mitigate any of these issues if they came up.

ANs.

Maintaining Data Integrity

1. **Validation and Verification**:
    - **Input Validation**: Ensure data meets specified criteria before processing.
    - **Verification**: Regular checks against benchmarks or standards.
2. **Access Controls and Auditing**:
    - **Role-Based Access Control (RBAC)**: Limit access based on user roles.
    - **Audit Trails**: Keep detailed logs of data access and modifications.
3. **Data Backup and Recovery**:
    - **Regular Backups**: Ensure data can be restored if corrupted or lost.
    - **Disaster Recovery Plans**: Strategy for data recovery post-catastrophe.
4. **Data Integrity Constraints**:
    - **Database Constraints**: Use primary keys, foreign keys, unique constraints.
    - **ACID Transactions**: Ensure transactions are Atomic, Consistent, Isolated, Durable.

Measuring Data Integrity

1. **Data Consistency Checks**:
    - **Cross-Verification**: Compare data across systems.
    - **Hashing**: Use cryptographic hashes to check for alterations.
2. **Error Rates**:
    - **Data Quality Metrics**: Measure errors like missing data, duplicates.
    - **Data Accuracy Rates**: Compare against correct values.
3. **Data Audits**:
    - **Regular Audits**: Systematic reviews for compliance.
    - **Automated Monitoring**: Continuous monitoring for anomalies.

Measuring Data Integrity

1. **Data Consistency Checks**:
    - **Cross-Verification**: Compare data across systems.
    - **Hashing**: Use cryptographic hashes to check for alterations.
2. **Error Rates**:
    - **Data Quality Metrics**: Measure errors like missing data, duplicates.
    - **Data Accuracy Rates**: Compare against correct values.
3. **Data Audits**:
    - **Regular Audits**: Systematic reviews for compliance.
    - **Automated Monitoring**: Continuous monitoring for anomalies.

Important Metrics

1. **Accuracy**: Correctness of data representing real-world values.
2. **Consistency**: Uniformity across datasets or systems.
3. **Completeness**: All required data is present.
4. **Timeliness**: Data is up-to-date and available when needed.

5. **Validity**: Data adheres to formats and rules.
6. **Uniqueness**: No duplicate records.

As a data engineer on the AWS cloud platform, ensuring data integrity is a cornerstone of my role.
Here's how I approach maintaining and measuring data integrity within AWS:

1. **AWS Data Governance Services**: I leveraged AWS services like AWS Lake Formation to establish and enforce data governance policies, access controls, and data cataloging.

2. **AWS Data Quality Tools**: I utilize AWS Glue for data discovery, cataloging, and ETL (Extract, Transform, Load) processes. Implement data quality checks within Glue jobs to ensure data accuracy and consistency.

3. **Amazon S3 Versioning**: We have enabled versioning on Amazon S3 buckets to track changes and revert to previous versions in case of data corruption or unintended modifications.

4. **AWS IAM Access Controls:** I implemented fine-grained access controls using AWS Identity and Access Management (IAM) to restrict access to data based on roles and permissions, ensuring data security and preventing unauthorized modifications.

5. **AWS CloudTrail Logging**: I have enabled AWS CloudTrail to log API activity and monitor data access, providing an audit trail for data integrity verification and compliance purposes.

Measuring Data Integrity:

1. **AWS CloudWatch Metrics**: I setted up CloudWatch alarms to monitor data quality metrics such as ETL job success rates, data validation errors, and latency in data pipelines.

2. **AWS Glue Data Catalog Metrics**: I used to monitor metrics within the Glue Data Catalog to track data lineage, table statistics, and metadata changes, ensuring data consistency and completeness.

3. **Amazon S3 Inventory:** I used Amazon S3 Inventory to generate reports on object metadata, enabling analysis of data storage patterns, access patterns, and identifying any anomalies that may affect data integrity.

4. **AWS CloudTrail Logs Analysis**: We used to analyze CloudTrail logs using Amazon Athena or Amazon Elasticsearch Service to detect any unauthorized access attempts or suspicious activity that may compromise data integrity.