HRG Data Engineer Test Assignment

1. Introduction

Users of Company X's numerous slot gaming applications can choose from a variety of games. The business sends structured JSON messages to an internal data bus via an event-driven architecture. Designing an effective Data Vault 2.0 model that can grow, store, and facilitate business intelligence reporting and analytics is the aim.

2. Chosen Technologies

- **Azure Synapse Analytics** – Big data transformations and analytics.
- **Azure Data Factory (ADF)** – Orchestration and ETL processing.
- **Databricks (Apache Spark)** – JSON parsing and processing.
- **Azure Key Vault** – Secure storage for sensitive data.
- **Snowflake** – Based on my experience, I chose Snowflake as an alternative warehouse because of its scalability for analytical queries.
- **Delta Lake** – Ensuring ACID compliance and efficient storage.
- **Azure Data Lake Storage (ADLS)** – Raw data storage.

3. Data Vault 2.0 Model

I structured the model using Data Vault 2.0 because it scales well and maintains historical tracking efficiently.

Hub Tables (Unique Business Entities)

1. **Hub_User**
    - user_hk (HASH) – Surrogate key
    - uid (STRING) – Business key (User ID)
    - load_dts (DATETIME) – Load timestamp
    - record_source (STRING) – Source system identifier
2. **Hub_Application**
    - app_hk (HASH) – Surrogate key
    - app (STRING) – Application name

- load_dts (DATETIME) – Load timestamp
- record_source (STRING) – Source system identifier

Link Tables (Relationships Between Entities)

1. **Link_User_Application**
    - user_app_hk (HASH) – Surrogate key
    - user_hk (HASH) – FK to Hub_User
    - app_hk (HASH) – FK to Hub_Application
    - load_dts (DATETIME) – Load timestamp
    - record_source (STRING) – Source system identifier

Satellite Tables (Descriptive and Historical Data)

1. **Sat_Auth_Event**
    - user_app_hk (HASH) – FK to Link_User_Application
    - email (STRING) – Nullable
    - phone (STRING) – Nullable
    - load_dts (DATETIME) – Load timestamp
    - record_source (STRING) – Source system identifier
2. **Sat_Spin_Event**
    - user_app_hk (HASH) – FK to Link_User_Application
    - spin_amount (INTEGER) – Spin value
    - publish_ts (DATETIME) – Event timestamp
    - load_dts (DATETIME) – Load timestamp
    - record_source (STRING) – Source system identifier
3. **Sat_Purchase_Event**
    - user_app_hk (HASH) – FK to Link_User_Application
    - purchase_amount (INTEGER) – Purchase value
    - publish_ts (DATETIME) – Event timestamp
    - load_dts (DATETIME) – Load timestamp

- record_source (STRING) – Source system identifier

Data Ingestion & Processing Steps Implemented

1. **Raw Data Storage:** JSON messages are stored in **ADLS** with a structured folder format:
    - raw/auth_events/YYYY/MM/DD/
    - raw/spin_events/YYYY/MM/DD/
    - raw/purchase_events/YYYY/MM/DD/
2. **Data Processing:**
    - **Databricks:** extracts, cleans, and validates JSON messages.
    - **Azure Data Factory:** orchestrates data movement into Azure Synapse/Snowflake.
    - **Delta Lake:** ensures efficient and ACID-compliant storage.
3. **Data Encryption:**

    User emails and phone numbers are encrypted using Azure Key Vault before storage.

Data Quality and governance tools Used:

- **Data Quality Checks**: Implement Great Expectations to validate data integrity.
- **Monitoring & Logging**: Use Azure Monitor and Datalog for pipeline observability.
- **Data Catalog & Governance**: Register metadata in Azure Purview.
- **Security & Access Control**: Enforce RBAC with Azure AD.

Challenges Faced

Managing Nested JSON Structures Difficulty	Before saving the pertinent properties in Delta Lake, they were effectively flattened and extracted using Apache Spark's built-in JSON methods.
Upholding Schema Evolution and Data Quality Challenge	Great Expectations was put into place for data validation, and Delta Lake's Schema Evolution was made able to dynamically adjust to new fields.
Providing Low-Latency Ingestion for Real-Time Processing Difficulty	To enhance Databricks query performance, Spark operations were optimized through the use of partitioning, bucketing, and caching.

	Databricks' Auto Loader for incremental ingestion was also taken into consideration.
Handling High-Scale Data Processing Costs Challenge	To effectively manage computing expenses, Photon-enabled clusters, task cluster termination policies, and Databricks optimized autoscaling were used.
Putting Secure Data Handling into Practice	Hashing was used to protect personally identifiable information and Azure Key Vault was used to securely manage encryption keys
Ensuring ACID Compliance and Integrity Challenge	Making sure that data in a high-volume streaming architecture is consistent and recoverable.
Effective Query Performance in Azure Synapse/Snowflake Issue	To improve performance, materialized views, result caching, and clustering strategies were put into practice.