HARD ROCK DIGITAL SYSTEM ANALYST TEST ASSIGNMENT

BY PREETIKA CHHOTRAY

The following assignment has been based on keeping Data Vault 2.0 technology in consideration.

DEVELOPING THE DATA MODEL

Starting with the description of the table structure, attribute composition and data types of the columns:

Based on Data Vault 2.0 data modeling technique, we have 3 main components:

1. Hubs
2. Links
3. Satellites

For the purpose of this assignment, I have considered my 3 hubs as mentioned below and the columns/data that they would contain:

i) Hub_Player
- Hub_player_key – is the hash key for Player table
- Player_id – acts as the business key for the table
- Load_timestamp – displays when the record was last added to the system

ii) Hub_App
- Hub_App_key – is the hash key for the Application table
- App_id – acts as the business key for the App table
- Load_timestamp – displays when the app record was last added to the system

iii) Hub_Event
- Hub_Event_key - is the hash key for Event table
- event_id – acts as the business key for the Event table
- Load_timestamp – displays when the event was last added to the system
- event_type – displays the specific type of event (i.e. auth, spins, purchase)

These Hub tables are connected to each other using Links.

The 3 Links (described with data) that I've considered are as follows:

i) Link_Player_Event (LPE)
- LPE_key – hash key for Link Player Event table
- LPE_id – Business key for the link relationship
- Player_id – connects to the data in Hub_Player
- Event_id – connects to the data in Hub_Event
- Load_timestamp – displays when the record was last added

ii) Link_Player_App (LPA)
- LPA_key – hash key for Link Player App table
- LPA_id – Business key for the link relationship
- Player_id – connects to the data in Hub_Player
- App_id - connects to the data in Hub_App
- Load_timestamp - displays when the record was last added

iii) Link_App_Event (LAE)
- LAE_key – hash key for Link Player Event table

- LAE_id – Business key for the link relationship
- App_id – connects to the data in Hub_App
- Event_id – connects to the data in Hub_Event
- Load_timestamp - displays when the record was last added

Each of the Hubs can have multiple Satellites associated to them, thus making it scalable without making much changes to the original model. For the purpose of this assignment, I have associated each Hub with 1 Satellite. The satellites are delineated as follows:

i) SAT_Player

- Hub_Player_key – hash key associated with the Hub_Player
- Player_id – business key for the Hub_Player
- Email – contains email info about the player
- Phone – contains phone number of the player
- Load_timestamp - displays when the record was last added

ii) SAT_App

- Hub_App_key – hash key associated with the Hub_App
- App_id – business key for Hub_App
- App_name – contains the name of the app
- App_type – contains the type of the app
- Load_timestamp – displays when the record was last added

iii) SAT_Event

- Hub_Event_key – hash key associated with the Hub_Event
- Event_id – business key for Hub_Event
- Load_timestamp – displays when the record was last added
- Payload – loads payload information

DEFINING THE DATA TYPES

The above columns will have the following data types:

COLUMN NAME	DATA TYPE
Player_id, Event_id	INTEGER
LPA_key, LPE_key, Hub_Player_key, Hub_App_key, Hub_Event_key, App_id, app_name, event_type, email, phone	VARCHAR
load_timestamp	TIMESTAMP
payload	JSON

IMPLEMENTATION OF THE DATA MODEL

To implement the above data model, the following technologies could be used to implement this:

1) DATA STORAGE – PostgreSQL is a great example if we want to setup out own server. It's great for smaller/mid-sized databases but for large databases RedShift could be used since it's a columnar warehouse.

2) MESSAGE BROKER/DATA PIPELINE – We can use **Apache Kafka** in which we rely on batch processing. As it is open source, we can set it up on docker or confluent.
3) ETL FRAMEWORK – Apache Airflow is great for complex workflows and can be used. Especially helpful because Airflow also has a UI to monitor DAGs in real-time, stores logs, thereby helping with debugging.
4) DEPLOYMENT – Docker is easy to use for deployment of PostGreSQL, **kafka,** etc. Another great tool is AWS (it's expensive but worth it).

Several other components that need to be developed to support the solution are as follows:

1. Data Ingestion Pipeline – To move events like spin, auth, purchase from source system to storage solution. KAFKA and PostgreSQL can be used for data storage and sending data to the event bus.
2. Transformation of data – Raw to Hub Loader would extract unique keys (e.g., player_id, application_id) from raw events and load them into Hubs and Link Table populator develop logic to map relationships between Hubs (e.g., Player to Event, Application to Event). Satellite Table Loader Extract descriptive and historical data (e.g., PII, event payloads) and populate Satellites.
3. Data Quality checks – checks like schema validation, schema duplication and ensuring foreign key relationships in the link tables are intact.
4. Data Governance/Security/Monitoring – masking PII data like phone number, email etc when not required, keeping audit logs to see who accessed what records at what time, creating role-based accesses to the database (can use RBAC by PostGreSQL). Datadog can be used for postGreSQL performance andto check logs of failure/error-handling.
5. Reporting/Analytics – The database can be connected to Tableau or Looker to provide cross-functional teams like Marketing team to have access to insights and dashboards.
6. Testing – Different methods of testing like Unit Testing, Integration testing and Load testing are recommended.