Data Storage Solution Proposal

1. Introduction

This document presents a storage solution designed to manage transactional data from multiple gaming applications while addressing important analytical needs. The proposed solution is based on the Data Vault 2.0 architecture, which ensures scalability, auditability, and the flexibility to adapt as requirements evolve. It includes a detailed explanation of the data model, the technologies chosen for implementation, and the additional components needed to support the system. This approach is designed to provide a reliable and future-proof foundation for managing and analyzing data effectively.

---

2. Proposed Technologies

Primary Database: PostgreSQL

- **Reason for Choice:**
    - PostgreSQL is highly reliable and supports advanced features like JSONB for semi-structured data, making it a great fit for transactional and analytical workloads.
    - Open-source, cost-effective, and widely adopted with a robust community.
    - Offers native support for partitioning, indexing, and time-series analysis, which suits analytical queries.

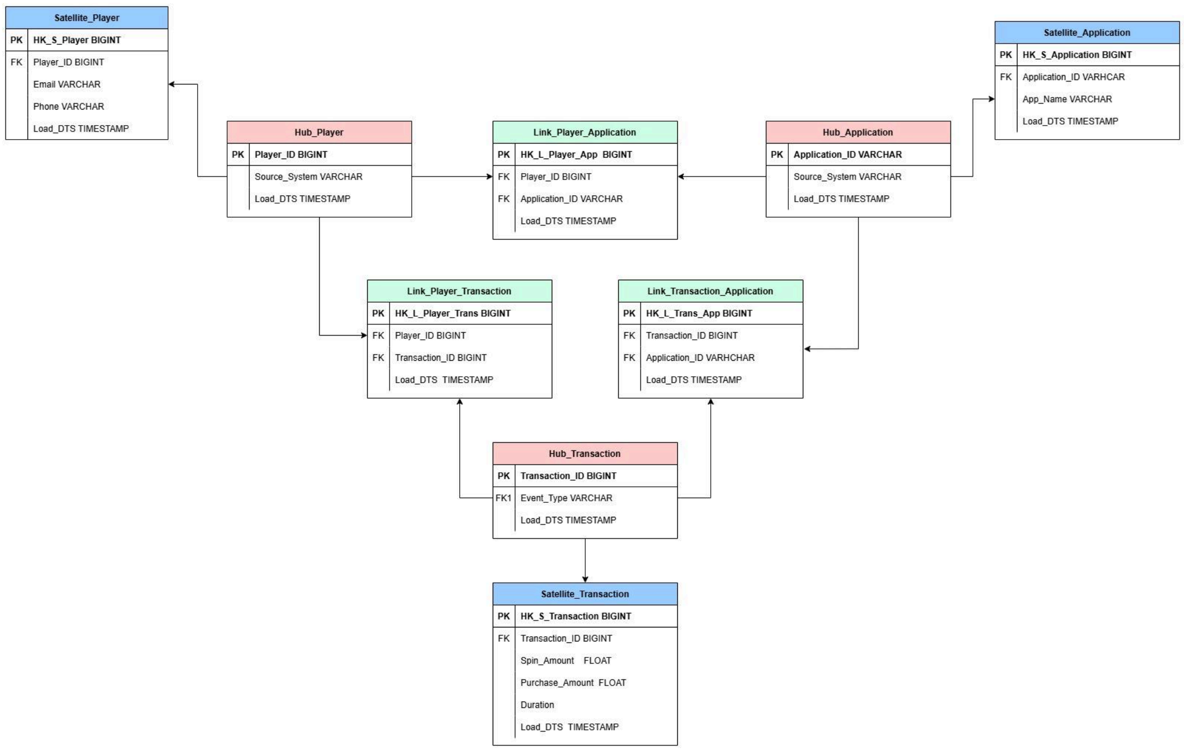
Alternatives: SQL Server

- While SQL Server could be used, PostgreSQL offers more flexibility and better support for advanced data types and extensions (e.g., TimescaleDB for time-series analysis).
- If the organization prefers SQL Server, schema definitions remain applicable with minor syntax adjustments.

---

3. Data Table Design

In this, HUB's are colored as Red, LINK's as Green and Satellite's as Blue.



Hub Tables

1. **Hub_Player**
   - **Purpose**: Stores unique identifiers for players with source information.
   - **Columns**:
     - `Player_ID` (Primary Key): Unique identifier for the player.
     - `Source_System`: Identifies the system of origin for the data stored in the hub.
       - For eg: Indicates if the player data came from a mobile app, a web app, or a third-party platform.
     - `Load_DTS`: Timestamp when the record was loaded.
2. **Hub_Application**
   - **Purpose**: Captures distinct application identifiers.
   - **Columns**:
     - `Application_ID` (Primary Key): Unique application identifier.
     - `Source_System`: Identifies the system of origin for the data stored in the hub.
       - For eg: Indicates whether the application metadata originated from an internal backend system or an external API.
     - `Load_DTS`: Timestamp when the record was loaded.

3. **Hub_Transaction**
   - **Purpose**: Contains unique transaction identifiers and types.
   - **Columns**:
     - `Transaction_ID` (Primary Key): Unique transaction identifier.
     - `Event_Type`: Type of event related to the transaction.
     - `Load_DTS`: Timestamp when the record was loaded.

Link Tables

1. **Link_Player_Application**
   - **Purpose**: Links players to applications they interact with.
   - **Columns**:
     - `HK_L_Player_App` (Primary Key): Hash key linking player and application.
     - `Player_ID` (Foreign Key): Reference to Hub_Player.
     - `Application_ID` (Foreign Key): Reference to Hub_Application.
     - `Load_DTS`: Timestamp when the record was loaded.
2. **Link_Player_Transaction**
   - **Purpose**: Links players to transactions they perform.
   - **Columns**:
     - `HK_L_Player_Trans` (Primary Key): Hash key linking player and transaction.
     - `Player_ID` (Foreign Key): Reference to Hub_Player.
     - `Transaction_ID` (Foreign Key): Reference to Hub_Transaction.
     - `Load_DTS`: Timestamp when the record was loaded.
3. **Link_Transaction_Application**
   - **Purpose**: Links transactions to associated applications.
   - **Columns**:
     - `HK_L_Trans_App` (Primary Key): Hash key linking transaction and application.
     - `Transaction_ID` (Foreign Key): Reference to Hub_Transaction.
     - `Application_ID` (Foreign Key): Reference to Hub_Application.
     - `Load_DTS`: Timestamp when the record was loaded.

Satellite Tables

1. **Satellite_Player**
   - **Purpose**: Holds detailed information about players.
   - **Columns**:
     - `HK_S_Player` (Primary Key): Hash key for player data.
     - `Player_ID` (Foreign Key): Reference to Hub_Player.
     - `Email`: Player's email.
     - `Phone`: Player's phone number.

        - `Load_DTS`: Timestamp when the record was loaded.

2. **Satellite_Application**
    - **Purpose**: Contains descriptive attributes of applications.
    - **Columns**:
        - `HK_S_Application` (Primary Key): Hash key for application data.
        - `Application_ID` (Foreign Key): Reference to Hub_Application.
        - `App_Name`: Name of the application.
        - `Load_DTS`: Timestamp when the record was loaded.
3. **Satellite_Transaction**
    - **Purpose**: Stores additional details about transactions.
    - **Columns**:
        - `HK_S_Transaction` (Primary Key): Hash key for transaction data.
        - `Transaction_ID` (Foreign Key): Reference to Hub_Transaction.
        - `Spin_Amount`: Amount involved in spins.
        - `Purchase_Amount`: Amount for purchases.
        - `Duration`: Duration of the transaction.
        - `Load_DTS`: Timestamp when the record was loaded.

---

4. Additional Components

1. Real-Time Data Integration Pipeline

- **Why Needed?** To ingest, transform, and load data into the Data Vault architecture in real-time to support use cases like real-time analytics and event-driven workflows.
- **Solution:**
    - Use **Apache Kafka** or **AWS Kinesis** to capture real-time events from applications, transactions, and player activities.
    - Create a **streaming ETL pipeline** using tools like **Apache Flink** or **Databricks** to process, clean, and enrich the data before writing it into the Data Vault tables in PostgreSQL.
    - Ensure schema evolution using **Debezium** for handling upstream schema changes seamlessly.

2. Intelligent Archival and Cost Optimization

- **Why Needed?** To ensure long-term storage is cost-efficient and compliant with data retention policies.
- **Solution:**
    - Develop an intelligent archival system that:
        - Identifies "cold" data (data not accessed in X days) using a query monitoring system.

        - Automatically migrates this data to cheaper storage like Amazon S3 or Azure Blob Storage.
        - Implements parquet or ORC formats for storage efficiency and faster query performance.
    - Build automation scripts to rehydrate archived data back into the database on demand.

3. Advanced Reporting and Analytics Framework

- **Why Needed?** To provide stakeholders with actionable insights directly from the Data Vault model without requiring a lot of technical know-how.
- **Solution:**
    - Use **dbt (Data Build Tool)** to transform Data Vault tables into business-oriented data marts (e.g., star or snowflake schemas) for analytics.
    - Integrate with **BI tools** like Tableau, Power BI, or Looker for advanced visualization.
    - Build custom APIs to serve data on-demand for applications or machine learning models.

4. Simulation and Testing Framework

- **Why Needed?** To test the scalability, robustness, and correctness of the system under real-world conditions.
- **Solution:**
    - Build a data simulator to generate synthetic player, application, and transaction data for testing.
    - Implement automated load testing using tools like JMeter to evaluate system performance under high transaction volumes.
    - Build a validation framework to ensure all data loads comply with schema and quality rules.

---

5. Conclusion

This Data Vault 2.0-based design ensures flexibility, scalability, and analytical robustness. The chosen technologies and components provide a solid foundation for current requirements while accommodating future growth.