Technologies Used to Implement the Storage Solution:

1. PostgreSQL (Event-Driven Data Storage)

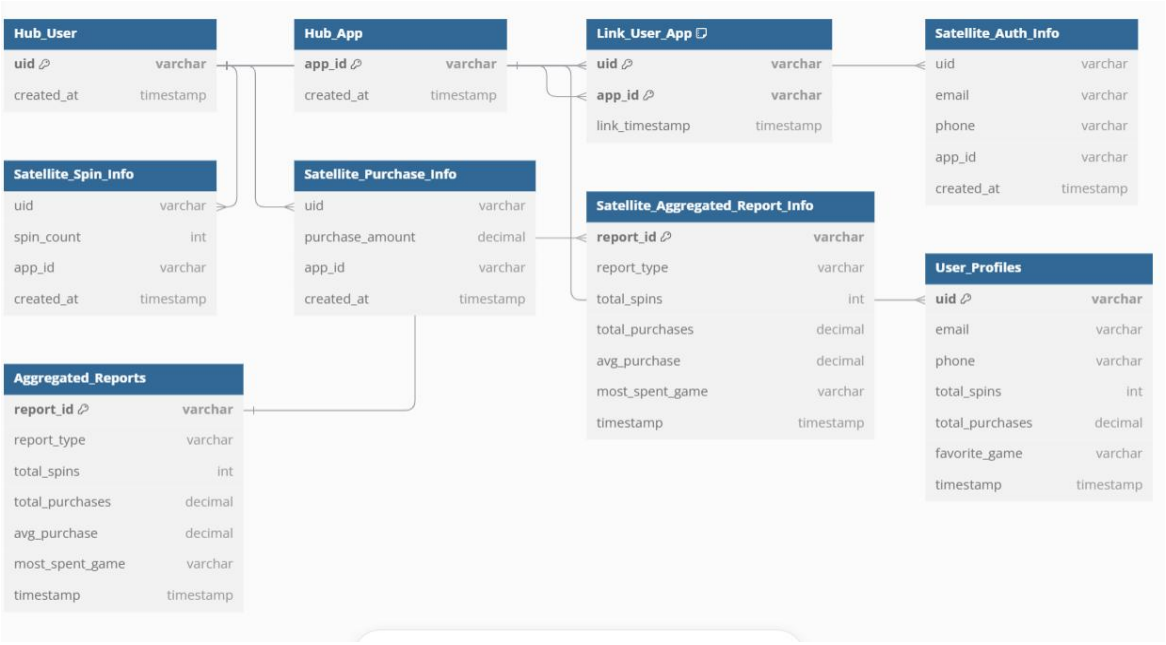
- PostgreSQL will be used for storing event-driven data, such as user authentication (auth), spins, and purchases.
- PostgreSQL is a highly reliable, ACID-compliant, and extensible relational database system that is well-suited for structured event-driven data storage with support for complex queries. It supports high-volume transaction processing and ensures consistency, making it ideal for storing events and associated metadata.

2. MySQL (User Profiles and Aggregated Data)

- MySQL will be used to store user profiles, aggregated reports, and other business insights based on events stored in PostgreSQL.
- MySQL is a fast and scalable relational database with a mature ecosystem. It provides high availability and is ideal for storing enriched data and aggregate reporting, making it suitable for creating analytics and business insights based on raw events stored in PostgreSQL.

Data Model Design

The **Data Vault 2.0 methodology** emphasizes scalability, flexibility, and agility. It involves three primary components: **Hubs**, **Links**, and **Satellites**. These elements work together to form a data model that allows for easy adaptation to changing business requirements.



1. Hubs (Business Keys)

Hubs contain the unique business keys. In this case, we have **users** and **applications** (games).

Hub_User (User Hub)

```
CREATE TABLE Hub_User (
   uid VARCHAR(255) PRIMARY KEY,
   created_at TIMESTAMP
);
```

- **Role**: Stores the unique identifier (uid) for each user.
- **Why Hub_User**: The uid uniquely identifies a user across different applications (games), serving as the central reference for all user-related data.

Hub_App (Application Hub)

```
CREATE TABLE Hub_App (
   app_id VARCHAR(255) PRIMARY KEY,
   created_at TIMESTAMP
);
```

- **Role**: Stores the unique identifier (app_id) for each application.
- **Why Hub_App**: The app_id uniquely identifies each game in the system.

2. Links (Relationships)

Links define the relationships between the business keys (Hubs). In this case, the relationship between **users** and **applications** is tracked.

Link_User_App (User-Application Link)

```
CREATE TABLE Link_User_App (
   uid VARCHAR(255),
   app_id VARCHAR(255),
   timestamp TIMESTAMP,
   PRIMARY KEY (uid, app_id)
);
```

- **Role**: Represents the relationship between users and applications. This link shows which user played which game at which point in time.
- **Why Link_User_App**: The link allows us to track user behavior across different applications over time.

3. Satellites (Descriptive Data)

Satellites store descriptive information that can change over time. This includes event-driven data such as authentication, spins, purchases, as well as enriched user data like profiles and reports.

Satellite_Auth_Info (Authentication Satellite)

```
CREATE TABLE Satellite_Auth_Info (
    uid VARCHAR(255),
    email VARCHAR(255),
    phone VARCHAR(255),
    app_id VARCHAR(255),
    created_at TIMESTAMP,
    FOREIGN KEY (uid) REFERENCES Hub_User(uid),
    FOREIGN KEY (app_id) REFERENCES Hub_App(app_id)
);
```

- **Role**: Stores user authentication data, including email, phone number, and app_id.
- **Why Satellite_Auth_Info**: Tracks authentication events, which may change over time (e.g., when a user changes their email or phone).

Satellite_Spin_Info (Spin Event Satellite)

```
CREATE TABLE Satellite_Spin_Info (
    uid VARCHAR(255),
    spin_count INTEGER,
    app_id VARCHAR(255),
    created_at TIMESTAMP,
    FOREIGN KEY (uid) REFERENCES Hub_User(uid),
    FOREIGN KEY (app_id) REFERENCES Hub_App(app_id)
);
```

- **Role**: Stores data related to spins (e.g., number of spins per user per game).
- **Why Satellite_Spin_Info**: Captures the spin events, which are key performance metrics for games.

Satellite_Purchase_Info (Purchase Event Satellite)

```
CREATE TABLE Satellite_Purchase_Info (
    uid VARCHAR(255),
    purchase_amount DECIMAL(10, 2),
    app_id VARCHAR(255),
    created_at TIMESTAMP,
    FOREIGN KEY (uid) REFERENCES Hub_User(uid),
    FOREIGN KEY (app_id) REFERENCES Hub_App(app_id)
);
```

- **Role**: Stores purchase event data, such as the amount spent by a user in each game.
- **Why Satellite_Purchase_Info**: Tracks purchases made by users, providing data for monetization and business analysis.

Satellite_Aggregated_Report_Info (Aggregated Reports Satellite)

```
CREATE TABLE Satellite_Aggregated_Report_Info (
    report_id VARCHAR(255) PRIMARY KEY,
    report_type VARCHAR(255),
    total_spins INTEGER,
    total_purchases DECIMAL(10, 2),
    avg_purchase DECIMAL(10, 2),
    most_spent_game VARCHAR(255),
    timestamp TIMESTAMP
);
```

- **Role**: Stores aggregated reports based on user activity.
- **Why Satellite_Aggregated_Report_Info**: Provides business insights, such as total spins, total purchases, and the most spent game for a specific time period.

MySQL Schema (User Profiles and Aggregated Data)

MySQL will store enriched user profiles and aggregated data.

User Profiles in MySQL

```
{
  "uid": "user123",
  "email": "user@example.com",
  "phone": "+1234567890",
  "total_spins": 1500,
  "total_purchases": 4500,
  "favorite_game": "app_3",
  "timestamp": "2024-10-12T17:30:00"
}
```

Aggregated Reports in MySQL

```
{
  "report_id": "report_001",
  "report_type": "weekly",
  "total_spins": 5000,
  "total_purchases": 12000,
  "avg_purchase": 2400,
  "most_spent_game": "app_3",
  "timestamp": "2024-10-12T17:30:00"
}
```

Data Flow Between PostgreSQL and MySQL

1. Real-Time Event Handling in PostgreSQL:

- **Event data** (authentication, spins, purchases) is ingested and stored in **PostgreSQL** in real-time. Each event is stored in the appropriate table based on the event type (authentication, spin, purchase), with msg_id as the primary key for each table.

2. Data Aggregation (ETL Process):

- A background ETL process (e.g., implemented using **Apache Spark** or **custom scripts**) aggregates event data from **PostgreSQL**. This aggregation includes:
    - Total spins, total purchases, and average purchases per user.
    - Most played games for each user.
- The aggregated data is then written to **MySQL** in the **user_profiles** table and **aggregated_reports** table.

3. User Profiles and Aggregated Reports in MySQL:

- **MySQL** stores the enriched **user profile data** and **aggregated reports** based on the events stored in **PostgreSQL**.
- These reports can be queried by business users to track metrics like average purchase per player, the most played games, etc.

4. Business Insights and Reporting:

- **MySQL** serves as the source for generating real-time business insights and reporting. Data can be exported to **business intelligence (BI) tools** (e.g., **Tableau** or **Power BI**) for visualizations.

Technology Stack

- **PostgreSQL**: A relational database management system for high-volume event data storage (auth, spins, purchases).
- **MySQL**: A relational database used for user profiles and aggregated reports.
- **ETL Process**: **Apache Spark** or **custom scripts** for periodic aggregation of event data.
- **API Layer**: REST or **GraphQL** API to query **MySQL** for user profiles and aggregated reports.
- **Monitoring**: **Prometheus** and **Grafana** to monitor the health of **PostgreSQL**, **MySQL**, and other components.
- **Log Aggregation**: **ELK Stack** (Elasticsearch, Logstash, Kibana) for event log collection and monitoring.

Additional Components Needed

1. Event Data Ingestion System:

- **Apache Kafka** to handle event streaming.
- **Kafka Consumer Services** to process events and store them in **PostgreSQL**.

2. ETL/Stream Processing:

- **Apache Spark** or **Apache Flink** for aggregating event data and updating **MySQL** with aggregated reports and user profiles.

3. User Profile Generation:

- Real-time or batch **user profile generation** stored in **MySQL** summarizing user interactions (purchases, spins).

4. Reporting Engine:

- A **REST API** or **GraphQL API** to query **MySQL** for real