# **Ievolve Event Management System - Database Design Document (DDD)**

## **1. Introduction**

### **1.1 Purpose**

The purpose of this Database Design Document (DDD) is to provide a detailed schema and structure for the database supporting the **Ievolve Event Management System**. This document defines the tables, columns, data types, relationships, and constraints necessary to store and manage all data related to accommodation for the **Phase 1: CM Trophy Accommodation Management** functionality.

### **1.2 Scope**

The scope of this document covers the logical and physical design of the database for the functionalities outlined in the Ievolve Event Management System SRS, Version 1.0. This includes tables for users (Admins, Coaches), hotels, participants (coaches, officials, players), and booking records, along with relevant audit trails. It does not cover database administration procedures, backup strategies, or detailed performance tuning, which would be addressed in separate operational documents.

### **1.3 Target Audience**

This document is intended for database administrators, back-end developers, system architects, and quality assurance engineers involved in the project.

### **1.4 Version History**

| **Version** | **Date** | **Author** | **Description of Changes** |
| --- | --- | --- | --- |
| 1.0 | August 6, 2025 | Vikram | Initial Draft of DDD. |

## **2. Overall Database Design**

The Ievolve Event Management System database will be built on **PostgreSQL**, leveraging its robust relational capabilities to ensure data integrity, consistency, and efficient querying. The design follows a normalized approach to minimize data redundancy and improve data quality.

The database will consist of the following primary tables:

* **users**: Stores authentication and profile data for Ievolve Admins and Team Coaches.
* **hotels**: Manages the inventory of hotels and their available rooms for specific date intervals.
* **participants**: Stores detailed information about all individuals (coaches, officials, players) attending events.
* **bookings**: Records the specific accommodation bookings made for participants, linking them to hotels and participants.
* **reassignments**: Tracks instances where participants are reassigned to different hotels.
* **audit\_log**: Records significant administrative actions for auditing purposes.

## **3. Entity Definitions (Tables)**

### **3.1 users Table**

* **Description:** Stores user authentication details and roles for system access.
* **Purpose:** To manage logins for Ievolve Admins (username/password + 2FA) and Team Coaches (mobile number + OTP).

| **Column Name** | **Data Type** | **Constraints** | **Description** |
| --- | --- | --- | --- |
| user\_id | VARCHAR(50) | PRIMARY KEY, NOT NULL, UNIQUE | Unique identifier (e.g., COA\_001, ADM\_001). |
| mobile\_number | VARCHAR(15) | UNIQUE, NULLABLE (for Admins) | Registered mobile for Team Coach login. |
| email | VARCHAR(100) | UNIQUE, NULLABLE (for Coaches) | Email for Ievolve Admin login. |
| password\_hash | VARCHAR(255) | NULLABLE (for Coaches using OTP only) | Hashed password for Ievolve Admins. |
| role | VARCHAR(20) | NOT NULL, CHECK ('admin', 'coach') | User's role in the system. |
| name | VARCHAR(100) | NOT NULL | Full name of the user. |
| otp\_secret | VARCHAR(255) | NULLABLE | Secret key for OTP generation (if applicable). |
| created\_at | TIMESTAMP | NOT NULL, DEFAULT CURRENT\_TIMESTAMP | Timestamp when the user record was created. |
| updated\_at | TIMESTAMP | NOT NULL, DEFAULT CURRENT\_TIMESTAMP | Timestamp of the last update to the user record. |

### **3.2 hotels Table**

* **Description:** Stores the inventory details for all hotels, including specific booking intervals.
* **Purpose:** To track total rooms, occupied rooms, and available rooms per hotel for defined date ranges.

| **Column Name** | **Data Type** | **Constraints** | **Description** |
| --- | --- | --- | --- |
| hotel\_id | VARCHAR(50) | NOT NULL | Unique identifier for the hotel (e.g., CHN001, trans\_CHN002). |
| instance\_code | INTEGER | NOT NULL | Identifier for a specific booking interval for the hotel. |
| hotel\_name | VARCHAR(100) | NOT NULL | Name of the hotel. |
| location | VARCHAR(100) | NOT NULL | General location of the hotel. |
| district | VARCHAR(50) | NOT NULL | District where the hotel is located. |
| address | VARCHAR(255) | NOT NULL | Full street address of the hotel. |
| pincode | VARCHAR(10) | NOT NULL | Postal code of the hotel. |
| start\_date | DATE | NOT NULL | Start date of the inventory availability interval. |
| end\_date | DATE | NOT NULL, CHECK (end\_date >= start\_date + INTERVAL '2 days') | End date of the inventory availability interval (min 3 days stay validation). |
| total\_rooms | INTEGER | NOT NULL, CHECK (total\_rooms >= 0) | Total number of rooms available for this interval. |
| occupied\_rooms | INTEGER | NOT NULL, DEFAULT 0, CHECK (occupied\_rooms <= total\_rooms) | Number of rooms currently occupied for this interval. (Represents sum for the interval) |
| available\_rooms | INTEGER | NOT NULL, DEFAULT 0, CHECK (available\_rooms >= 0) | Number of rooms currently available for this interval. (Represents sum for the interval) |
| created\_at | TIMESTAMP | NOT NULL, DEFAULT CURRENT\_TIMESTAMP | Timestamp when the record was created. |
| updated\_at | TIMESTAMP | NOT NULL, DEFAULT CURRENT\_TIMESTAMP | Timestamp of the last update. |
| PRIMARY KEY (hotel\_id, instance\_code) |  |  |  |
| UNIQUE (hotel\_id, start\_date, end\_date) |  |  |  |

### **3.3 participants Table**

* **Description:** Stores details for all players, coaches, and officials.
* **Purpose:** To manage individual participant information and their assigned bookings.

| **Column Name** | **Data Type** | **Constraints** | **Description** |
| --- | --- | --- | --- |
| participant\_id | VARCHAR(50) | PRIMARY KEY, NOT NULL, UNIQUE | Unique ID (e.g., COA\_001, PLA\_001, OFC\_001). |
| role | VARCHAR(20) | NOT NULL, CHECK ('coach', 'player', 'official') | Role of the participant. |
| name | VARCHAR(100) | NOT NULL | Full name of the participant. |
| mobile\_number | VARCHAR(15) | NULLABLE | Mobile number of the participant. |
| discipline | VARCHAR(50) | NOT NULL | Sport discipline (e.g., 'Football', 'Athletics'). |
| district | VARCHAR(50) | NULLABLE (for coaches/officials if not applicable) | District of the participant. |
| team\_name | VARCHAR(100) | NULLABLE (for coaches/officials) | Name of the team for players. |
| coach\_id | VARCHAR(50) | NULLABLE, FOREIGN KEY REFERENCES users(user\_id) | Links player to their coach. |
| hotel\_id | VARCHAR(50) | NOT NULL, FOREIGN KEY REFERENCES hotels(hotel\_id) | The ID of the hotel assigned to this participant. |
| booking\_reference\_number | VARCHAR(100) | NOT NULL, UNIQUE | Unique booking reference (e.g., R\_PLA\_001\_BOOKING, TPRE\_COA\_002\_TRANSIT). |
| booking\_start\_date | DATE | NOT NULL | Scheduled check-in date. |
| booking\_end\_date | DATE | NOT NULL | Scheduled check-out date. |
| checkin\_status | VARCHAR(20) | NOT NULL, DEFAULT 'pending', CHECK ('pending', 'checked-in', 'checked-out') | Current check-in status. |
| checkin\_time | TIMESTAMP | NULLABLE | Actual check-in timestamp. |
| checkout\_time | TIMESTAMP | NULLABLE | Actual check-out timestamp. |
| checkout\_initiator | VARCHAR(20) | NULLABLE, CHECK ('admin', 'coach') | Indicates if early checkout was initiated by 'admin' or 'coach'. |
| created\_at | TIMESTAMP | NOT NULL, DEFAULT CURRENT\_TIMESTAMP | Timestamp when the record was created. |
| updated\_at | TIMESTAMP | NOT NULL, DEFAULT CURRENT\_TIMESTAMP | Timestamp of the last update. |
| TravelPOC | VARCHAR(15) | NULLABLE | Mobile number of the Travel POC |

### **3.4 bookings Table (Conceptual - actual booking status managed in participants)**

* **Description:** This table is conceptual for now, as the core booking status and reference are managed directly within the participants table. However, if a more complex booking system were to evolve (e.g., group bookings, payment tracking), this table would be expanded. For Phase 1, it primarily serves to link participants to specific hotel instances and track the overall booking status.
* **Purpose:** To represent a confirmed booking slot for a participant at a specific hotel instance.

| **Column Name** | **Data Type** | **Constraints** | **Description** |
| --- | --- | --- | --- |
| booking\_id | VARCHAR(100) | PRIMARY KEY, NOT NULL, UNIQUE | Matches participants.booking\_reference\_number. |
| participant\_id | VARCHAR(50) | NOT NULL, FOREIGN KEY REFERENCES participants(participant\_id) | Links to the participant who has this booking. |
| hotel\_id | VARCHAR(50) | NOT NULL | Hotel ID for this booking. |
| hotel\_instance\_code | INTEGER | NOT NULL | Instance code of the hotel for this booking. |
| booking\_type | VARCHAR(20) | NOT NULL, CHECK ('regular', 'pre\_transit', 'post\_transit', 'reassigned') | Type of booking. |
| status | VARCHAR(20) | NOT NULL, DEFAULT 'booked', CHECK ('booked', 'cancelled', 'completed') | Overall status of the booking. |
| created\_at | TIMESTAMP | NOT NULL, DEFAULT CURRENT\_TIMESTAMP | Timestamp when the booking record was created. |
| updated\_at | TIMESTAMP | NOT NULL, DEFAULT CURRENT\_TIMESTAMP | Timestamp of the last update to the booking record. |
| FOREIGN KEY (hotel\_id, hotel\_instance\_code) REFERENCES hotels(hotel\_id, instance\_code) |  |  |  |

### **3.5 reassignments Table**

* **Description:** Records details of hotel reassignments for audit purposes.
* **Purpose:** To maintain a clear audit trail of participants moved from one hotel to another.

| **Column Name** | **Data Type** | **Constraints** | **Description** |
| --- | --- | --- | --- |
| reassignment\_id | SERIAL | PRIMARY KEY | Unique identifier for each reassignment event. |
| original\_booking\_ref | VARCHAR(100) | NOT NULL, FOREIGN KEY REFERENCES participants(booking\_reference\_number) | The original booking reference number. |
| new\_booking\_ref | VARCHAR(100) | NOT NULL, UNIQUE, FOREIGN KEY REFERENCES participants(booking\_reference\_number) | The new booking reference number for the reassigned group. |
| reassigned\_by\_user\_id | VARCHAR(50) | NOT NULL, FOREIGN KEY REFERENCES users(user\_id) | The Admin who performed the reassignment. |
| reassignment\_date | TIMESTAMP | NOT NULL, DEFAULT CURRENT\_TIMESTAMP | Timestamp of the reassignment. |
| reason | TEXT | NULLABLE | Reason for the reassignment (e.g., "hotel commitment compromised"). |
| notes | TEXT | NULLABLE | Any additional notes about the reassignment. |

### **3.6 audit\_log Table**

* **Description:** General purpose audit log for significant system actions, including data edits.
* **Purpose:** To track critical operations performed by Admins (e.g., early checkouts, data uploads, and record modifications), ensuring traceability and accountability.

| **Column Name** | **Data Type** | **Constraints** | **Description** |
| --- | --- | --- | --- |
| log\_id | SERIAL | PRIMARY KEY | Unique identifier for each log entry. |
| action\_type | VARCHAR(50) | NOT NULL | Type of action (e.g., 'data\_upload', 'early\_checkout\_admin', 'record\_edit', 'record\_delete'). |
| action\_by\_user\_id | VARCHAR(50) | NOT NULL, FOREIGN KEY REFERENCES users(user\_id) | User who performed the action. |
| target\_entity | VARCHAR(50) | NOT NULL | The table/entity affected (e.g., 'participants', 'hotels'). |
| target\_id | VARCHAR(100) | NOT NULL | The primary key/identifier of the specific record affected. |
| timestamp | TIMESTAMP | NOT NULL, DEFAULT CURRENT\_TIMESTAMP | When the action occurred. |
| reason | TEXT | NULLABLE | The reason provided by the Admin for performing the action (e.g., for edits). |
| details | JSONB | NULLABLE | JSON object containing additional action details, such as old\_value and new\_value for edited fields. |

### **3.7 Audit Log Business Logic**

The audit\_log table is specifically crafted to capture and track significant administrative actions, ensuring transparency and accountability within the system.

**What Triggers an Audit Log Entry?**

An entry in the audit\_log table is triggered whenever an **Ievolve Admin** performs a critical action that modifies or impacts core data. Based on our SRS and DDD, these actions include:

* **Data Uploads:** When an Admin uploads the Hotel Inventory Sheet, Coach and Official Data Sheet, or Player Data Sheet.
* **Record Edits:** When an Admin uses the separate UI to modify any existing record (e.g., in hotels or participants tables).
* **Record Deletions:** When an Admin deletes any existing record.
* **Admin-Initiated Early Checkouts:** When an Admin sets an early checkout date for a team.
* **Hotel Reassignments:** When an Admin creates a new booking record for a reassigned group.
* **Transit Check-in/Check-out Management:** When an Admin manages the check-in or check-out status for transit bookings.

**How is the Audit Information Captured?**

When one of the above actions occurs, the application's backend business logic will perform the following steps:

* **Identify the Action:** The system determines the specific action\_type (e.g., 'data\_upload', 'record\_edit', 'early\_checkout\_admin', 'hotel\_reassignment').
* **Identify the Actor:** The user\_id of the **Ievolve Admin** currently logged in and performing the action is captured and stored in action\_by\_user\_id.
* **Identify the Target:**
  + The target\_entity (e.g., 'hotels', 'participants', 'team') is identified.
  + The target\_id (the unique identifier of the specific record or group of records affected) is captured. For a single record edit, it would be the participant\_id or hotel\_id. For a bulk action like an upload or team early checkout, it might be a team identifier or a relevant booking\_reference\_number.
* **Capture Timestamp:** The timestamp is automatically recorded at the moment the action is performed.
* **Capture Reason (for Edits/Specific Actions):** For actions like record edits or reassignments, the Admin will be prompted to provide a reason for the change. This reason is then stored in the reason column.
* **Capture Details (Old vs. New Values):** This is crucial for edits. For a 'record\_edit' action\_type, the details column (which is a JSONB data type) will store a JSON object containing:
  + old\_value: A snapshot of the record's relevant fields *before* the change.
  + new\_value: A snapshot of the record's relevant fields *after* the change.
  + This allows for a complete "before and after" view of any modifications. For other action types (like 'data\_upload'), details might contain file names, row counts, or summary information.

**Database Storage:**

All this information is then inserted as a new row into the audit\_log table.

This comprehensive approach ensures that every significant administrative change is meticulously recorded, providing a robust audit trail for review, compliance, and troubleshooting.

## **4. Relationships**

The relationships between the tables are critical for maintaining data integrity and enabling complex queries.

* **users to participants**:
  + users.user\_id (PK) -> participants.coach\_id (FK)
  + Type: One-to-Many (One coach can manage many players).
* **hotels to participants**:
  + hotels.hotel\_id (PK) -> participants.hotel\_id (FK)
  + Type: One-to-Many (One hotel can be assigned to many participants).
  + *Note: The instance\_code is part of hotels composite PK, but participants only references hotel\_id. The specific instance for a booking would be inferred by matching dates or explicitly stored if needed for very granular tracking.*
* **participants to bookings (Conceptual Link):**
  + participants.booking\_reference\_number (PK) -> bookings.booking\_id (PK)
  + Type: One-to-One (Each participant's booking reference is a unique booking).
* **participants to reassignments**:
  + participants.booking\_reference\_number (PK) -> reassignments.original\_booking\_ref (FK)
  + participants.booking\_reference\_number (PK) -> reassignments.new\_booking\_ref (FK)
  + Type: One-to-Many (One original booking can have multiple reassignments, and a new booking is a result of one reassignment).
* **users to reassignments**:
  + users.user\_id (PK) -> reassignments.reassigned\_by\_user\_id (FK)
  + Type: One-to-Many (One admin can perform many reassignments).
* **users to audit\_log**:
  + users.user\_id (PK) -> audit\_log.action\_by\_user\_id (FK)
  + Type: One-to-Many (One user can perform many audited actions).

## **5. Derived/Calculated Metrics and Business Logic for Reporting**

This section describes how key metrics, particularly estimated room vacancies, are calculated by the application layer using the underlying database design. These are not directly stored as columns for every single day but are derived on demand for reporting purposes.

### **5.1 Estimated Daily Room Vacancy Calculation**

To determine the estimated vacant rooms for a specific HotelID on a specific\_date, the application will perform the following steps:

1. **Retrieve Hotel Capacity for the Day:**
   * Query the hotels table to find the total\_rooms for all instance\_code records where the specific\_date falls within the start\_date and end\_date range of the given HotelID.
   * Sum the total\_rooms from these relevant instances to get the total estimated room capacity for that hotel on that specific day.
2. **Identify Participants Staying on the Day:**
   * Query the participants table to find all individuals (players, coaches, officials) whose hotel\_id matches the target HotelID.
   * Filter these participants to include only those whose booking\_start\_date is less than or equal to the specific\_date AND whose booking\_end\_date is greater than or equal to the specific\_date.
   * Exclude any participants marked with a checkin\_status of 'pending' or 'checked-out' (unless the checkout date is after the specific\_date for early checkouts).
3. **Estimate Rooms Consumed:**
   * For each participant identified in Step 2, apply the following room consumption ratios:
     + **Players:** 1 room per 3 players (i.e., each player consumes 1/3 of a room).
     + **Team Coaches:** 1 room per 2 team coaches (i.e., each coach consumes 1/2 of a room).
     + **Officials:** 1 room per 1 official (i.e., each official consumes 1 room).
   * Sum these fractional room consumptions for all relevant participants.
   * Round the total sum of consumed rooms **up** to the nearest whole number to get the estimated occupied rooms for that day.
4. **Calculate Estimated Vacancies:**
   * Subtract the Estimated Rooms Consumed (from Step 3) from the Total Hotel Capacity for the Day (from Step 1).
   * The result is the Estimated Daily Vacant Rooms.

This calculation provides an administrative estimate and acknowledges that actual room assignments are handled by the hotel, outside the system's direct tracking.

### **5.2 Early Checkout Tracking and Reporting**

This section details how the system will track and report on early checkouts, differentiating between those initiated by an Admin and those initiated by a Team Coach.

1. **Tracking Initiator Type:**
   * The participants table will include a checkout\_initiator column (VARCHAR(20), NULLABLE, CHECK ('admin', 'coach')).
   * When a Team Coach performs an early checkout (FR-2.3 in SRS), the application will update the checkout\_time for the relevant players and set checkout\_initiator to 'coach'.
   * When an Ievolve Admin initiates an early checkout (FR-2.7 in SRS), the application will update the checkout\_time for the relevant participants and set checkout\_initiator to 'admin'.
2. **Reporting and Summaries:**
   * **Date-wise Early Checkouts:** The system can query the participants table, filter by checkout\_time within a specific date range, and group the results by checkout\_initiator to show counts for 'admin' vs. 'coach' initiated checkouts.
   * **Overall Summary:** A similar query without a date filter can provide total counts for each initiator type.
   * **Detailed Audit Trail:** The audit\_log table (as described in Section 3.7) will provide granular details for each early checkout action, including who initiated it, when, the reason (for Admin-initiated), and the specific participants affected. This complements the summary data from the participants table.

## **6. Indexing Strategy**

To optimize query performance, especially for filtering and reporting, the following indexes will be considered:

* **users**:
  + mobile\_number (Unique Index)
  + email (Unique Index)
  + role (B-tree Index)
* **hotels**:
  + hotel\_name (B-tree Index)
  + district (B-tree Index)
  + start\_date, end\_date (Composite B-tree Index for date range queries)
* **participants**:
  + booking\_reference\_number (Unique Index)
  + coach\_id (B-tree Index)
  + hotel\_id (B-tree Index)
  + checkin\_status (B-tree Index)
  + discipline (B-tree Index)
  + district (B-tree Index)
  + name (B-tree Index for search)
  + mobile\_number (B-tree Index for search)
  + checkout\_initiator (B-tree Index for reporting)
* **reassignments**:
  + original\_booking\_ref (B-tree Index)
  + new\_booking\_ref (Unique Index)
* **audit\_log**:
  + action\_type (B-tree Index)
  + action\_by\_user\_id (B-tree Index)
  + timestamp (B-tree Index)

## **7. Security Considerations (Database Level)**

* **Least Privilege:** Database users and application roles will be configured with the minimum necessary permissions (read, write, update, delete) on specific tables.
* **Data Encryption:** Sensitive data at rest (e.g., password\_hash, potentially mobile numbers if required by policy) will be encrypted using PostgreSQL's native encryption features or application-level encryption.
* **Secure Connections:** All connections to the database will use SSL/TLS encryption.
* **Input Sanitization:** The application layer will perform rigorous input validation and sanitization to prevent SQL injection attacks.
* **Auditing:** The audit\_log table will capture critical actions, providing a trail for security reviews.