The revised database structure is designed to ensure normalization, reduce redundancy, and improve data integrity. Here's a detailed explanation of each collection and their relationships:

Collections

Visitors Collection

Purpose: To store the basic information about each visitor.

Schema:
```json

{
  "_id": ObjectId("..."),
  "name": "John Doe",
  "phone_number": "1234567890"
}

```

Fields:
- _id: Unique identifier for each visitor (automatically generated by MongoDB).
- name: Name of the visitor.
- phone_number: Contact number of the visitor.

Visitor Sessions Collection

Purpose: To store details of each visit session, including the primary visitor, purpose, and check-in/out details.

Schema:
```json

{
  "_id": ObjectId("..."),
  "visitor_id": ObjectId("..."), // Reference to visitors collection
  "purpose_of_visit": "Campus Tour",
  "entry_gate": "Gate 1",
  "check_in_time": ISODate("2024-06-26T09:00:00Z"),
  "exit_gate": "Gate 1",
  "check_out_time": ISODate("2024-06-26T17:00:00Z"),
```

```
  "group_size": 4,
  "group_id": ObjectId("...") // Reference to visitor_groups collection
}

```

Fields:
- _id: Unique identifier for each session (automatically generated by MongoDB).
- visitor_id: Reference to the primary visitor in the visitors collection.
- purpose_of_visit: Reason for the visit.
- entry_gate: Gate where the visitor entered.
- check_in_time: Timestamp when the visitor checked in.
- exit_gate: Gate where the visitor exited.
- check_out_time: Timestamp when the visitor checked out.
- group_size: Number of people in the group (including the primary visitor).
- group_id: Reference to the visitor_groups collection for group details.

Visitor Groups Collection

Purpose: To manage group visit details, linking to the main visitor session.

Schema:
```json

{
  "_id": ObjectId("..."),
  "session_id": ObjectId("..."), // Reference to visitor_sessions collection
  "group_members": [
    {
      "card_id": 101,
      "check_in_time": ISODate("2024-06-26T09:00:00Z"),
      "check_out_time": ISODate("2024-06-26T16:00:00Z")
    },
    {
      "card_id": 102,
      "check_in_time": ISODate("2024-06-26T09:00:00Z"),
      "check_out_time": ISODate("2024-06-26T16:30:00Z")
    }
    // Add more group members if needed
  ]
}

```

Fields:
- _id: Unique identifier for each group session (automatically generated by MongoDB).
- session_id: Reference to the visitor session in the visitor_sessions collection.
- group_members: Array of group member details, each with:
  - card_id: Visitor card ID assigned to the group member.
  - check_in_time: Timestamp when the group member checked in.
  - check_out_time: Timestamp when the group member checked out.

Visitor Cards Collection

Purpose: To manage the status and assignment of visitor cards.

Schema:
```json

{
  "_id": ObjectId("..."),
  "card_id": 101,
  "status": "in use", // available, in use, used
  "assigned_to": ObjectId("...") // Reference to visitor_sessions collection or null
}

```

Fields:
- _id: Unique identifier for each card (automatically generated by MongoDB).
- card_id: ID of the visitor card.
- status: Current status of the card (available, in use, used).
- assigned_to: Reference to the visitor_sessions collection if the card is currently assigned, otherwise null.

Users Collection

Purpose: To manage login information and roles for security and admin personnel.

Schema:
```json


{
```

```
  "_id": ObjectId("..."),
  "username": "security1",
  "password_hash": "hashedpassword",
  "role": "security" // Can be 'security' or 'admin'
}
```

Fields:
- _id: Unique identifier for each user (automatically generated by MongoDB).
- username: Username for the security or admin personnel.
- password_hash: Hashed password for secure authentication.
- role: Role of the user (security or admin).

Improved Normalization and Redundancy Removal

By splitting the visitor sessions and group details into separate collections, we avoid redundancy and ensure each piece of information is stored only once. Here's how the improved design handles different scenarios:

Handling Different Scenarios

Scenario 1: Individual Check-In

Front End: Security personnel enters visitor details and manually assigns an ID card.
Backend:
- Register the visitor in the visitors collection if not already present.
- Create a new session in the visitor_sessions collection.
- Update the corresponding card in visitor_cards to mark it as "in use".

Scenario 2: Group Check-In

Front End: Security personnel enables "Group Entry," enters primary visitor details, and manually assigns ID cards to the group members.
Backend:
- Register the primary visitor in the visitors collection if not already present.
- Create a new session in the visitor_sessions collection.
- Create a new group entry in the visitor_groups collection, linked to the session.

- Update the corresponding cards in visitor_cards to mark them as "in use".

Scenario 3: Individual Check-Out

Front End: Security personnel enters or scans the visitor card ID.
Backend:
- Update the check_out_time in the visitor_sessions collection.
- Update the corresponding card in visitor_cards to mark it as "used".

Scenario 4: Primary Visitor Checks Out but Group Members Remain

Front End: Security personnel enters or scans the primary visitor's card ID.
Backend:
- Update the check_out_time for the primary visitor in the visitor_sessions collection.
- If group members are still checked in, their status remains unchanged.

Scenario 5: Group Member Checks Out

Front End: Security personnel enters or scans the group member's card ID.
Backend:
- Update the check_out_time for the specific group member in the visitor_groups collection.
- Update the corresponding card in visitor_cards to mark it as "used".

This database structure ensures efficient handling of both individual and group visits, while maintaining clear and organized data relationships across collections.