Jahhaazzz

MUHAMMAD AHMAD (BSCS24068)
MOHIB ABDUL KARIM (BSCS24052)
MARYAM ARSHAD (BSCS24052)

1. Schema Design

Jahhaazzz, is not just flight booking or basic Airport Management System. It includes airport infrastructure, airlines, aircrafts, flight scheduling, passengers, cargo, runway management, staff assignments, maintenance, incidents, and even flight consolidation.

So after listing all real-world entities involved in our airport system, we separated them into logical groups:

- ➢ Airport infrastructure (Airport, Terminals, Gates, Runways)
- ➢ Airline and aircraft management
- ➢ Flight and scheduling system
- ➢ Passenger and ticketing system
- ➢ Cargo management
- ➢ Staff and task management
- ➢ Operational exceptions (incidents, consolidation)

2. Data Dictionary (Tables, Columns, Data Types & Constraints)

2.1 Airport

Stores airport details.

Columns:

- ★ airport_id (int) – Primary Key
- ★ airport_name (varchar(100))
- ★ city (varchar(100))
- ★ country (varchar(100))
- ★ airport_code (varchar(10))
- ★ timezone (varchar(50))
- ★ created_at (timestamp)

Constraints:

➔ PRIMARY KEY (airport_id)
➔ UNIQUE (airport_code)

Why UNIQUE on airport_code?

Because airport_code (like LHR, ISB, KHI) must be unique in real aviation systems. airport_id is for the sake of internal record

2.2 Airline

Stores airline information.

Columns:

★ airline_id (int) – Primary Key
★ airline_name (varchar(100))
★ country (varchar(100))
★ airline_code (varchar(10))
★ created_at (timestamp)

Constraints:

➔ PRIMARY KEY (airline_id)
➔ UNIQUE (airline_code)

Similar to Airport, airline_code is a business identifier.

2.3 Aircraft_types

This table stores the type/model of aircraft. We separated this because many aircrafts can belong to the same type.

Columns:

★ aircraft_type_id (int) – Primary Key
★ model_name (varchar(100))
★ manufacturer (varchar(100))
★ seat_capacity (int)

Constraints:

➔ PRIMARY KEY on aircraft_type_id
➔ seat_capacity should logically be positive (business rule)

Reason:
We separated aircraft type to avoid repeating model and manufacturer details for every aircraft. This is done to satisfy the 3rd Normal form.

2.4 Aircraft

Stores individual aircrafts.

Columns include:

- ★ aircraft_id (int, auto increment) – Primary Key
- ★ airline_id (int) – Foreign Key
- ★ aircraft_type_id (int) – Foreign Key
- ★ registration_number (varchar(50)) – UNIQUE
- ★ status (ENUM)
- ★ seat counts (economy, business, first)
- ★ maintenance dates
- ★ current_airport (FK)

Constraints:

- ➔ PRIMARY KEY (aircraft_id)
- ➔ FOREIGN KEY (airline_id) → Airline
- ➔ FOREIGN KEY (aircraft_type_id) → Aircraft_types
- ➔ FOREIGN KEY (current_airport) → Airport
- ➔ UNIQUE (registration_number)

Reason:
We separated aircraft from aircraft type to maintain normalization.

2.5 Aircraft_Maintenance

This table stores maintenance history of aircraft.

Columns:

- ★ maintenance_id (int)
- ★ aircraft_id (int)
- ★ maintenance_date (date)
- ★ maintenance_type (ENUM: Inspection, Repair, Engine Check, Cleaning)
- ★ description (text)
- ★ technical_issues_found (int)
- ★ status (ENUM: Scheduled, Completed, Cancelled)

Constraints:

- ➔ PRIMARY KEY (maintenance_id)
- ➔ FOREIGN KEY (aircraft_id) references Aircraft(aircraft_id)

We separated this because maintenance is a repeating event.
If we stored maintenance details inside the Aircraft table, it would violate 1NF and 3NF.

2.6 Terminals

This table stores all terminals inside an airport.
One airport can have multiple terminals(1 to M relationship)

Columns:

- ★ terminal_id (int)
- ★ airport_id (int)
- ★ terminal_name (varchar(50))
- ★ terminal_code (varchar(10))
- ★ created_at (timestamp)

Constraints:

- ➔ PRIMARY KEY (terminal_id)
- ➔ FOREIGN KEY (airport_id) references Airport(airport_id)

This separation avoids repeating terminal data inside the airport table, helping maintain normalization.

2.7 Gates

This table stores gates inside terminals.

Columns:

- ★ gate_id (int)
- ★ terminal_id (int)
- ★ gate_number (varchar(10))
- ★ status (ENUM: active, maintenance)
- ★ created_at (timestamp)

Constraints:

- ➔ PRIMARY KEY (gate_id)
- ➔ FOREIGN KEY (terminal_id) references Terminals(terminal_id)

Relationship:

We separated gates because gates depend on terminals, not directly on airports.

2.8 Runways

This table stores runways for each airport.

Columns:

★ runway_id (int)
★ airport_id (int)
★ runway_code (varchar(20))
★ length (int)
★ status (ENUM: active, maintenance)
★ created_at (timestamp)

Constraints:

➔ PRIMARY KEY (runway_id)
➔ FOREIGN KEY (airport_id) references Airport(airport_id)

2.9 Runway_Bookings

Stores scheduled runway usage before actual operation.

Columns:

★ booking_id (int)
★ runway_id (int)
★ aircraft_id (int)
★ booking_date (date)
★ start_time (time)
★ end_time (time)
★ booking_status (ENUM: Reserved, Approved, Cancelled, Completed)
★ created_at (timestamp)

Constraints:

➔ PRIMARY KEY (booking_id)
➔ FOREIGN KEY (runway_id) references Runways(runway_id)
➔ FOREIGN KEY (aircraft_id) references Aircraft(aircraft_id)

This prevents double booking logic issues and supports scheduling system.

2.10 Flight_Runway_Usage

Stores actual runway usage during landing or takeoff.

Columns:

★ usage_id (int)
★ runway_id (int)
★ flight_schedule_id (int)
★ operation_type (ENUM: Landing, Takeoff)
★ actual_time (datetime)

Constraints:

➔ PRIMARY KEY (usage_id)
➔ FOREIGN KEY (runway_id) references Runways(runway_id)
➔ FOREIGN KEY (flight_schedule_id) references
➔ Flight_Schedules(flight_schedule_id)

We separated booking from actual usage to maintain data accuracy and normalization.

2.11 Flights

Represents route definition (not actual scheduled flight).

Columns:

★ flight_id (int) – PK
★ airline_id (FK)
★ source_airport_id (FK)
★ destination_airport_id (FK)
★ flight_type (ENUM)
★ estimated_duration

Constraints:

➔ PRIMARY KEY (flight_id)
➔ FOREIGN KEY → Airline
➔ FOREIGN KEY → Airport (source & destination)

Reason:
Flight is route-level data. Actual departure times are stored in Flight_schedules.

2.12 Flight_schedules

Stores actual scheduled instance of a flight.

Columns:

★ flight_schedule_id – PK
★ flight_id – FK
★ aircraft_id – FK

★ departure_datetime
★ arrival_datetime
★ gate_id – FK
★ flight_status (ENUM)

Constraints:

➔ PRIMARY KEY
➔ FOREIGN KEYS
➔ Trigger validation: arrival must be after departure

Reason:
We separated schedules because one flight route can run multiple times on different dates.

2.13 Passengers

Stores passenger personal information.

Columns:

★ passenger_id (int)
★ first_name (varchar(100))
★ last_name (varchar(100))
★ date_of_birth (date)
★ passport_number (varchar(50))
★ nationality (varchar(100))
★ phone_number (varchar(20))
★ email (varchar(100))
★ created_at (timestamp)

Constraints:

➔ PRIMARY KEY (passenger_id)
➔ UNIQUE (passport_number)

2.14 Tickets

Represents booking of passenger on specific flight schedule.

Columns:

★ ticket_id – PK
★ passenger_id – FK

★ flight_schedule_id – FK
★ seat_number
★ seat_class (Economy, Business, First)
★ status

Constraints:

➔ PRIMARY KEY
➔ FOREIGN KEYS
➔ UNIQUE (flight_schedule_id, seat_number)

Reason:
Ensures no two passengers get same seat in same flight.

2.15 Baggage

Stores baggage information for passengers.

Columns:

★ baggage_id (int)
★ ticket_id (int)
★ flight_schedule_id (int)
★ weight (decimal)
★ baggage_type (ENUM: Checked, Carry-on, Cargo)
★ status (ENUM: Checked-In, Loaded, In Transit, Unloaded, Lost)
★ created_at (timestamp)

Constraints:

➔ PRIMARY KEY (baggage_id)
➔ FOREIGN KEY (ticket_id) references Tickets(ticket_id)
➔ FOREIGN KEY (flight_schedule_id) references
➔ Flight_Schedules(flight_schedule_id)

We separated baggage from tickets to avoid multi-valued dependency (4NF improvement).

2.16 Boarding_Records

Stores boarding activity of passengers.

Columns:

★ boarding_id (int)
★ ticket_id (int)

- ★ flight_schedule_id (int)
- ★ boarding_time (datetime)
- ★ boarding_status (ENUM: Pending, Boarded, Denied)
- ★ gate_id (int)
- ★ created_at (timestamp)

Constraints:

- ➔ PRIMARY KEY (boarding_id)
- ➔ FOREIGN KEY (ticket_id) references Tickets(ticket_id)
- ➔ FOREIGN KEY (flight_schedule_id) references
- ➔ Flight_Schedules(flight_schedule_id)
- ➔ FOREIGN KEY (gate_id) references Gates(gate_id)

This separation allows tracking real boarding events without altering ticket data.

2.17 Staff

Stores airport staff.

Columns:

- ★ staff_id (int)
- ★ first_name (varchar(100))
- ★ last_name (varchar(100))
- ★ phone_number (varchar(20))
- ★ email (varchar(100))
- ★ role (ENUM: 'ground_staff', 'security', 'maintenance', 'operations_manager', 'admin')
- ★ hire_date (date)
- ★ salary (decimal(10,2))
- ★ airport_id (int)
- ★ status (ENUM: 'active', 'on_leave', 'terminated')

Constraints:

- ➔ PRIMARY KEY (staff_id)
- ➔ FOREIGN KEY (airport_id)

2.18 Shifts

This table stores working shifts of staff members.
It helps us manage availability and assignment of staff on specific dates and times.

Columns:

- ★ shift_id (int)
- ★ staff_id (int)
- ★ shift_date (date)
- ★ shift_start (time)
- ★ shift_end (time)
- ★ availability_status (ENUM: Available, Assigned, Off)

Constraints:

- ➔ PRIMARY KEY (shift_id)
- ➔ FOREIGN KEY (staff_id) references Staff(staff_id)

One staff member can have many shifts, but one shift belongs to only one staff member.

2.19 Tasks & Task_Assignments

Examples:

- ❖ Boarding supervision
- ❖ Gate management
- ❖ Baggage handling
- ❖ Runway clearance
- ❖ Security inspection

Each task is linked to one flight schedule.

Columns:

- ★ task_id (int)
- ★ flight_schedule_id (int)
- ★ task_type (varchar(100))
- ★ task_description (text)
- ★ priority_level (ENUM: 'low', 'medium', 'high')
- ★ status (ENUM: 'pending', 'in_progress', 'completed')
- ★ created_at (timestamp)

Constraints:

- ➔ PRIMARY KEY (task_id)
- ➔ FOREIGN KEY (flight_schedule_id) → Flight_Schedules(flight_schedule_id)

2.20 Task_Assignments

This table resolves the many-to-many relationship between Staff and Tasks.

Columns:

★ assignment_id (int)
★ task_id (int)
★ staff_id (int)
★ assigned_at (timestamp)
★ assignment_status (ENUM: 'assigned', 'completed')

Constraints:

➔ PRIMARY KEY (assignment_id)
➔ FOREIGN KEY (task_id) → Tasks(task_id)
➔ FOREIGN KEY (staff_id) → Staff(staff_id)
➔ UNIQUE (task_id, staff_id)

2.21 Cargo

Stores cargo shipment information linked to flights.

Columns:

★ cargo_id (int)
★ flight_id (int)
★ tracking_number (varchar(10))
★ cargo_type (ENUM: General, Perishable, Hazardous, Fragile, Live Animals, Mail)
★ description (text)
★ weight_kg (decimal)
★ origin_airport_id (int)
★ destination_airport_id (int)
★ sender_name (varchar(50))
★ sender_contact (varchar(50))
★ receiver_name (varchar(50))
★ receiver_contact (varchar(50))
★ status (ENUM: Booked, Loaded, In Transit, Unloaded, Customs Hold, Delivered, Cancelled)
★ is_insured (boolean)

Constraints:

➔ PRIMARY KEY (cargo_id)
➔ UNIQUE (tracking_number)
➔ FOREIGN KEY (flight_id) references Flights(flight_id)
➔ FOREIGN KEY (origin_airport_id) references Airport(airport_id)
➔ FOREIGN KEY (destination_airport_id) references Airport(airport_id)

Tracking number must be unique because it identifies shipment globally.

2.22 Flight_Consolidation

Stores merging of two flight schedules.

Columns:

- ★ consolidation_id (int)
- ★ original_flight_schedule_id (int)
- ★ new_flight_schedule_id (int)
- ★ reason (varchar(200))
- ★ consolidation_date (datetime)

Constraints:

- ➔ PRIMARY KEY (consolidation_id)
- ➔ FOREIGN KEY (original_flight_schedule_id) references
- ➔ Flight_Schedules(flight_schedule_id)
- ➔ FOREIGN KEY (new_flight_schedule_id) references
- ➔ Flight_Schedules(flight_schedule_id)

This prevents data loss and keeps history of merged flights.

2.23 Flight_Incidents

Stores incidents related to flight schedules.

Columns:

- ★ incident_id (int)
- ★ flight_schedule_id (int)
- ★ incident_type (ENUM: Technical Issue, Weather Delay, Crew Issue, Security Issue, Emergency)
- ★ description (text)
- ★ reported_at (timestamp)
- ★ resolved_at (timestamp)
- ★ flight_status (ENUM: Open, Resolved)

Constraints:

- ➔ PRIMARY KEY (incident_id)
- ➔ FOREIGN KEY (flight_schedule_id) references
- ➔ Flight_Schedules(flight_schedule_id)

This allows incident tracking without modifying flight table directly.

2.24 Payments

Stores payments for tickets and cargo.

Columns:

- ★ payment_id (int)
- ★ ticket_id (int)
- ★ cargo_id (int)
- ★ amount (decimal)
- ★ payment_method (ENUM: Credit Card, Cash, Online Transfer)
- ★ payment_status (ENUM: Pending, Completed, Failed)
- ★ payment_date (timestamp)

Constraints:

- ➔ PRIMARY KEY (payment_id)
- ➔ FOREIGN KEY (ticket_id) references Tickets(ticket_id)
- ➔ FOREIGN KEY (cargo_id) references Cargo(cargo_id)

We allow either ticket_id or cargo_id depending on payment type.

2.25 Crew_Requirements

Stores required staff roles for each flight schedule.

Columns:

- ★ requirement_id (int)
- ★ flight_schedule_id (int)
- ★ role_required (varchar(100))
- ★ number_required (int)

Constraints:

- ➔ PRIMARY KEY (requirement_id)
- ➔ FOREIGN KEY (flight_schedule_id) references
- ➔ Flight_Schedules(flight_schedule_id)

This avoids storing crew counts inside flight schedule and keeps design flexible.

3. Relationship Cardinality Table

#	Parent Table	Child Table	Cardinality	Explanation
1	Airport	Terminals	1 : Many	One airport has multiple terminals
2	Airport	Runways	1 : Many	One airport has multiple runways
3	Airport	Staff	1 : Many	One airport employs many staff
4	Airport	Flights (source)	1 : Many	One airport can be departure for many flights
5	Airport	Flights (destination)	1 : Many	One airport can be arrival for many flights
6	Airport	Cargo (origin)	1 : Many	One airport can send many cargo shipments
7	Airport	Cargo (destination)	1 : Many	One airport can receive many cargo shipments
8	Airline	Aircraft	1 : Many	One airline owns many aircraft
9	Airline	Flights	1 : Many	One airline operates many flights
10	Aircraft_types	Aircraft	1 : Many	One aircraft type can have multiple aircraft
11	Aircraft	Aircraft_Maintenance	1 : Many	One aircraft can have many maintenance records
12	Aircraft	Runway_Bookings	1 : Many	One aircraft can book runways multiple times

13	Aircraft	Flight_schedules	1 : Many	One aircraft can operate many schedules
14	Terminals	Gates	1 : Many	One terminal has multiple gates
15	Gates	GateAssignments	1 : Many	One gate can be assigned many times
16	Flight_schedules	GateAssignments	1 : Many	One schedule may have multiple gate assignments
17	Runways	Runway_Bookings	1 : Many	One runway can have many booking slots
18	Runways	Flight_Runway_Usage	1 : Many	One runway used by many schedules
19	Flights	Flight_schedules	1 : Many	One flight operates on many dates
20	Flight_schedules	Tickets	1 : Many	One schedule has many tickets
21	Passengers	Tickets	1 : Many	One passenger can book many tickets
22	Tickets	Baggage	1 : Many	One ticket can have multiple baggage items
23	Tickets	Boarding_Records	1 : 1	One ticket produces one boarding record per schedule
24	Tickets	Payments	1 : Many	One ticket may have multiple payments
25	Flights	Cargo	1 : Many	One flight carries many cargo shipments
26	Cargo	Payments	1 : Many	One cargo shipment may have multiple payments
27	Staff	Shifts	1 : Many	One staff member can have many shifts
28	Staff	Task_Assignments	1 : Many	One staff member can be assigned many tasks

29	Tasks	Task_Assignments	1 : Many	One task can be assigned to many staff
30	Flight_schedules	Tasks	1 : Many	One schedule generates multiple tasks
31	Flight_schedules	Boarding_Records	1 : Many	One schedule has many boarding records
32	Flight_schedules	Crew_Requirements	1 : Many	One schedule has multiple crew requirements
33	Flight_schedules	Flight_Incidents	1 : Many	One schedule may have multiple incidents
34	Flight_schedules	Flight_Runway_Usage	1 : Many	One schedule can have landing & takeoff usage records
35	Flight_schedules	Flight_Consolidation (original)	1 : Many	One schedule may be merged into others
36	Flight_schedules	Flight_Consolidation (new)	1 : Many	One schedule may receive multiple merged schedules

3. 1 Many-to-Many Relationships (Resolved)

#	Relationship	Type	Resolved By
1	Staff <-> Tasks	Many : Many	Task_Assignments
2	Flight_schedules <-> Runways	Many : Many (logical)	Flight_Runway_Usage
3	Flight_schedules <-> Gates	Many : Many (logical)	GateAssignments

4. Index Specifications with Rationale

We added indexes mainly for performance on frequently searched columns.

Examples:

- Index on Flights(airline_id)
  Because flights are often filtered by airline.
- Index on Flight_schedules(departure_datetime)
  Because searching upcoming flights by date is common.
- Index on Tickets(passenger_id)
  Because we often search all bookings of a passenger.
- Index on Staff(role)
  Because staff assignment queries filter by role.

We did not index every column because over-indexing reduces insert/update performance.

5. Normalization Overview

During schema design:

1NF:
We ensured no repeating groups and atomic values.

2NF:
We avoided composite key partial dependency by mostly using surrogate primary keys.

3NF:
We separated Airline, Airport, Aircraft_types to remove transitive dependencies.

BCNF:
For every functional dependency, the determinant is a super key, separating entities like Aircraft_types, Airlines, and Airports to remove transitive dependencies.

4NF:
We separated Tasks and Task_Assignments to remove multi-valued dependencies.

5NF:
We decomposed many-to-many relationships properly to avoid join dependency redundancy.

So overall, the schema is normalized up to 3NF strictly, and practically satisfies 4NF/5NF conditions as well

6. User Roles Definition

We defined different user roles for system control.

1. Admin
   - Full access
   - Can create airports, airlines, aircrafts

    - Manage staff and schedules
    - View all reports
2. Operations Manager
    - Manage flight schedules
    - Assign gates and runways
    - Handle incidents
    - Assign tasks to staff
3. Ticketing Staff
    - Create passengers
    - Issue tickets
    - Check-in passengers
    - Manage boarding records