ACID Documentation

Mohammad Ahmad – BSCS24068
Mohib Abdul Karim – BSCS24058
Maryam Arshad – BSCS24052

In Jahhaazzz, there are many operations happening at the same time, ticket booking, baggage check-in, runway booking, staff assignment, payments, etc.

4 important transaction scenarios from our system are described below as per requirement.

1. Ticket Booking + Payment Transaction

Scenario

The following flow occurs when a passenger books a ticket:

Ticket record is created

↓

Seat availability is checked

↓

Seat count is reduced

↓

Payment record is inserted

↓

Boarding eligibility becomes valid

All of this must either happen fully or not happen at all.

Atomicity

If for any unforeseeable reason, the payment fails, the ticket will not be confirmed Same goes for the seats. If the ticket assignment fails, a seat should not be allocated.

So either everything commits, or everything rolls back.

We use database transactions (BEGIN → COMMIT / ROLLBACK).

Consistency

Some constraints are implemented on database level to ensure the data remains consistent and valid

- Seat count cannot go below zero
- Ticket must reference valid flight_schedule
- Payment must reference valid ticket

All foreign keys and check constraints ensure this.

Isolation

Two passengers should not book the same last seat at the same time.

We use:

Isolation Level: REPEATABLE READ

Reason:

- Prevents dirty reads
- Prevents non-repeatable reads
- Prevents lost updates

This ensures seat count remains correct.

Durability

Once a ticket is booked and payment is confirmed, data is permanently stored in the servers even if a system failure occurs or crashes.

Pre-logging the database ahead-of-time ensures durability.

2. Flight Delay Update Transaction

Scenario

If a flight schedule status changes to Delayed:

1. Flight_schedules status updated
2. GateAssignments may be updated
3. Runway_Bookings timing may change
4. Staff shift timings may be adjusted

5. Passengers notified (application level)

All must stay consistent.

Atomicity

Delay update must not partially update runway but not gate.

Either all updates succeed or all revert.

Consistency

- New departure time must be valid
- Runway slot must not overlap
- Staff shifts must not violate working hours

Constraints + triggers ensure logical consistency.

Isolation

Multiple admins should not edit same flight schedule simultaneously.

Isolation Level: READ COMMITTED

Reason:

- Prevents dirty reads
- Allows good performance
- Acceptable for admin updates

Durability

Once a delay is committed, the system permanently stores new timing.

3. Runway Booking Transaction

Scenario

When runway is booked for takeoff/landing:

1. Runway_Bookings record inserted
2. Conflict check for overlapping time slots
3. Flight_Runway_Usage record inserted

Atomicity

If conflict exists, nothing should be inserted.

Consistency

- No two bookings overlap on same runway
- Booking must reference valid runway
- Must reference valid aircraft or flight schedule

This is done through unique time-slot constraint, referential keys and safety triggers

Isolation

Isolation Level: SERIALIZABLE

Runway scheduling is critical. Two parallel bookings must not both pass conflict check.

Durability

Once a runway slot is confirmed, it must never disappear after crash.

4. Staff Task Assignment Transaction

Scenario

When assigning staff to a flight task:

1. Check staff availability in Shifts
2. Insert Task_Assignment
3. Update availability_status

Atomicity

If the staff is not available, no assignment should happen.

Consistency

- Staff must exist
- Task must exist
- Same staff cannot be assigned twice to same task (unique constraint)

Isolation

Isolation Level: READ COMMITTED

Reason:
We only need to prevent dirty reads.
Small overlap risk is acceptable since backend checks availability.

Concurrency Strategy Used in Jahhaazzz

We use:

1. Pessimistic locking for:
    - Seat booking
    - Runway booking
2. Optimistic approach for:
    - Admin updates
    - Task assignments
3. Foreign key constraints to prevent orphan data
4. Unique constraints to prevent duplication
5. Triggers only for validation, not heavy logic

Note:

Complex business logic (like flight reconsolidation) is handled in the backend, not in triggers.