Railway Reservation & Management Portal

Team Members:
S Sitaraman (200953080), Sakshi Arjun (220953594), Pratham Singh (220953612)

Schema Reduction & Normalization from Entity Relationship Diagram:

Entities:

1. **User**(user_id, user_name, user_email, user_password, user_sex, user_phone, user_dob, user_created_at, user_is_admin)
2. **Passenger**(passenger_id, passenger_name, passenger_email, passenger_sex, passenger_phone, passenger_dob)
3. **Ticket**(ticket_id, food_preference, created_at, status)
4. **Payment**(payment_id, status, price, created_at)
5. **Seat**(seat_id, type, price)
6. **Train**(train_id, train_name, source, destination, start_time, end_time, status, total_seats)
7. **Station**(station_id, station_name, city)

Relationships:

1. **buy_ticket**(user_id, ticket_id)
2. **purchase**(ticket_id, payment_id)
3. **allocated**(ticket_id, seat_id)
4. **travel**(user_id, passenger_id)
5. **administrating**(user_id, is_admin)
6. **having**(train_id, seat_id)
7. **schedule**(train_id, station_id, platform, arrival_time, departure_time)

Reduction Steps:

We will start reduction assuming all the strong entity relations are retained.

1. **buy_ticket**(user_id, ticket_id,)
   We do not require a separate table for this relation since it's full participation from ticket entity into the user entity and also has a many-to-one relationship. We can simply add user_id into the ticket schema to remove this redundancy.
2. **purchase**(ticket_id, payment_id)
   We do not require a separate table for this relation since it's a one-to-one relation from ticket entity into the payment entity. We can either add ticket_id into payment schema or add payment_id into the ticket schema to remove this redundancy.
3. **allocated**(ticket_id, seat_id)

We do not require a separate table for this relation since it’s a one-to-one relation from ticket entity into the seat entity. We can either add ticket_id into seat schema or add seat_id into ticket schema to remove this redundancy.

4. **travel**(user_id, passenger_id)
   Since this is an identifying relationship between strong entity User and weak entity Passenger without any descriptive attribute, we can choose to ignore this relation by adding user_id into the Passenger schema and making the combination of user_id and passenger_id as the primary key in the Passenger schema.
5. **administrating**(user_id, is_admin)
   This is a self-relation on the User entity. It has only one attribute, is_admin, which we can incorporate into the User schema.
6. **having**(seat_id, train_id)
   We do not require a separate table for this relation since the relation from seat entity to the train entity is a many-to-one relationship. We can simply add train_id into the seat schema to remove this redundancy.
7. **schedule**(train_id, station_id, platform, arrival_time, departure_time)
   This relation is between train and station with three descriptive attributes. We will retain this relation with train_id and station_id as the primary key.

Final schema after reduction:

1. **User**(user_id, user_name, user_email, user_password, user_sex, user_phone, user_dob, user_created_at, user_is_admin)
   This schema will be retained (Strong Entity)
2. **Ticket**(ticket_id, user_id *[FK],* payment_id *[FK]*, seat_id *[FK]*, food_preference, created_at, status)
   This schema will be retained (Strong Entity)
3. **Payment**(payment_id, status, price, created_at)
   This schema will be retained (Strong Entity)
4. **Seat**(seat_id, train_id *[FK]*, type, price)
   This schema will be retained (Strong Entity)
5. **Train**(train_id, train_name, source, destination, start_time, end_time, status, total_seats)
   This schema will be retained (Strong Entity)
6. **Station**(station_id, station_name, city)
   This schema will be retained (Strong Entity)
7. **Passenger**(user_id *[FK]*, passenger_id, passenger_name, passenger_email, passenger_sex, passenger_phone, passenger_dob)
   Weak Entity schema retained with user_id & passenger_id as primary key
8. **Schedule**(train_id *[FK]*, station_id *[FK]*, platform, arrival_time, departure_time)
   Retained relation after reduction

Normalisation

UniversalRelation = {user_id, user_name, user_email, user_password, user_sex, user_phone, user_dob, user_created_at, passenger_id, passenger_name, passenger_email, passenger_sex, passenger_phone, passenger_dob, ticket_id, food_preference, ticket_created_at, ticket_status, payment_id, payment_status, payment_price, payment_created_at, seat_id, seat_type, seat_price, train_id, train_name, train_source, train_destination, train_start_time, train_end_time, train_status, train_total_seats, station_id, station_name, station_city, platform, arrival_time, departure_time, is_admin}

Functional Dependencies:

A) user_id → name, email, password, sex, phone, dob, created_at, is_admin
B) ticket_id → user_id, payment_id, seat_id, food_preference, created_at, status
C) payment_id → status, price, created_at
D) seat_id → train_id, type, price
E) train_id → train_name, source, destination, start_time, end_time, status, total_seats
F) station_id → station_name, city
G) user_id, passenger_id → passenger_id, passenger_name, passenger_email, passenger_sex, passenger_phone, passenger_dob
H) train_id, station_id → platform, arrival_time, departure_time

Taking closure to determine the possible candidate keys and, eventually, the primary keys for the schemas

1. user_id+ = { user_id, name, email, password, sex, phone, dob, created_at, is_admin} (Not a candidate key)
2. ticket_id+ = { ticket_id, user_id, payment_id, seat_id, food_preference, created_at, status, name, email, password, sex, phone, dob, created_at, is_admin} (Not a candidate key)
3. (user_id,ticket_id,payment_id,seat_id,train_id,station_id,passenger_id,station_id)+ = R

Hence, **(user_id,ticket_id,payment_id,seat_id,train_id,station_id,passenger_id,station_id)** is a candidate key for the relation R.

Normalizing to First Normal Form (1NF):

A relational schema R is in first normal form if the domains of all attributes in relation R are atomic, i.e., disallows composite attributes, multivalued attributes, and nested relations.

R = {user_id, user_name, user_email, user_password, user_sex, user_phone, user_dob, user_created_at, passenger_id, passenger_name, passenger_email, passenger_sex, passenger_phone, passenger_dob, ticket_id, food_preference, ticket_created_at, ticket_status, payment_id, payment_status, payment_price, payment_created_at, seat_id, seat_type, seat_price, train_id, train_name, train_source, train_destination, train_start_time, train_end_time, train_status, train_total_seats, station_id, station_name, station_city, platform, arrival_time, departure_time, is_admin}

Since all attributes in our functional dependencies are atomic in nature, relation R follows the First Normal Form (1NF).

Normalizing to Second Normal Form (2NF):

2NF is based on the concept of Full Functional Dependency. A relation schema R is in 2NF if it is in 1NF form, and every non-prime attribute A in R is fully functionally dependent on the primary key of R.

R = {user_id, user_name, user_email, user_password, user_sex, user_phone, user_dob, user_created_at, passenger_id, passenger_name, passenger_email, passenger_sex, passenger_phone, passenger_dob, ticket_id, food_preference, ticket_created_at, ticket_status, payment_id, payment_status, payment_price, payment_created_at, seat_id, seat_type, seat_price, train_id, train_name, train_source, train_destination, train_start_time, train_end_time, train_status, train_total_seats, station_id, station_name, station_city, platform, arrival_time, departure_time, is_admin}

Relation R is already in 1NF as all our elements are indivisible units (atomic).

Analyzing Functional Dependencies:

FD1 (user_id → name, email, password, sex, phone, dob, created_at, is_admin)
Relational schema R does not satisfy 2NF as all non-prime attributes are not fully functionally dependent on the primary key of R.

We will split R into two relations, R1 and R2

R1 - {user_id, user_name, user_email, user_password, user_sex, user_phone, user_dob, user_created_at, user_is_admin}
R1 is in 2NF since it is fully functionally dependent, and only the primary key (*user_id*) defines all other non-prime attributes

R2 - {user_id, passenger_id, passenger_name, passenger_email, passenger_sex, passenger_phone, passenger_dob, ticket_id, food_preference, ticket_created_at, ticket_status, payment_id, payment_status, payment_price, payment_created_at, seat_id, seat_type, seat_price, train_id, train_name, train_source, train_destination, train_start_time, train_end_time, train_status, train_total_seats, station_id, station_name, station_city, platform, arrival_time, departure_time}

Relational schema R2 does not satisfy 2NF as all non-prime attributes are not fully functionally dependent on the primary key of R2.

We will split R2 into two relations R2 and R3 according to FD2.
FD2 (ticket_id → user_id, payment_id, seat_id, food_preference, created_at, status)

R2 - {ticket_id, user_id, payment_id, seat_id, food_preference, created_at, status}
R2 is in 2NF since it is fully functionally dependent and only primary key (ticket_id) defines all other non-prime attributes

R3 - {user_id, payment_id, payment_status, payment_price, payment_created_at, seat_id, seat_type, seat_price, train_id, train_name, train_source, train_destination, train_start_time, train_end_time, train_status, train_total_seats, station_id, station_name, station_city, platform, arrival_time, departure_time, passenger_id, passenger_name, passenger_email, passenger_sex, passenger_phone, passenger_dob}

Relational schema R3 does not satisfy 2NF as all non-prime attributes are not fully functionally dependent on the primary key of R3.

We will split R3 into two relations R3 and R4 according to FD3
FD3 (payment_id → status, price, created_at)

R3 - {payment_id, status, price, created_at}
R3 is in 2NF since it is fully functionally dependent and only primary key (*payment_id*) defines all other non-prime attributes

R4 - {user_id, seat_type, seat_price, train_id, train_name, train_source, train_destination, train_start_time, train_end_time, train_status, train_total_seats, station_id, station_name, station_city, platform, arrival_time, departure_time, passenger_id, passenger_name, passenger_email, passenger_sex, passenger_phone, passenger_dob}

Relational schema R4 does not satisfy 2NF as all non-prime attributes are not fully functionally dependent on the primary key of R4.

We will split R4 into two relations R4 and R5 according to FD4
FD4 (seat_id → train_id, type, price)

R4 - {seat_id, train_id, type, price}
R4 is in 2NF since it is fully functionally dependent and only primary key (*seat_id*) defines all other non-prime attributes

R5 - {user_id, ticket_id, train_id, train_name, train_source, train_destination, train_start_time, train_end_time, train_status, train_total_seats, station_id, station_name, station_city, platform, arrival_time, departure_time, passenger_id, passenger_name, passenger_email, passenger_sex, passenger_phone, passenger_dob}

Relational schema R5 does not satisfy 2NF as all non-prime attributes are not fully functionally dependent on the primary key of R5.

We will split R5 into two relations R5 and R6 according to FD5
FD5 (train_id → train_name, source, destination, start_time, end_time, status, total_seats)

R5 - {train_id, train_name, source, destination, start_time, end_time, status, total_seats}
R5 is in 2NF since it is fully functionally dependent and only primary key (*train_id*) defines all other non-prime attributes

R6 - {user_id, ticket_id, train_id, station_id, station_name, station_city, platform, arrival_time, departure_time, passenger_id, passenger_name, passenger_email, passenger_sex, passenger_phone, passenger_dob}

Relational schema R6 does not satisfy 2NF as all non-prime attributes are not fully functionally dependent on the primary key of R6.

We will split R6 into two relations R6 and R7 according to FD6
FD6(station_id → station_name, city)

R6 - {station_id, station_name, city}
R6 is in 2NF since it is fully functionally dependent and only primary key (*station_id*) defines all other non-prime attributes

R7 - {user_id, train_id, station_id, platform, arrival_time, departure_time, passenger_id, passenger_name, passenger_email, passenger_sex, passenger_phone, passenger_dob}

Relational schema R7 does not satisfy 2NF as all non-prime attributes are not fully functionally dependent on the primary key of R7.

We will split R7 into two relations R7 and R8 according to FD7
FD7(user_id, passenger_id → passenger_name, passenger_email, passenger_sex, passenger_phone, passenger_dob)

R7 - {user_id, passenger_id, passenger_name, passenger_email, passenger_sex, passenger_phone, passenger_dob}
R7 is in 2NF since it is fully functionally dependent and only primary key (*station_id*) defines all other non-prime attributes

R8 - {train_id, station_id, platform, arrival_time, departure_time}
Relational schema R8 automatically satisfies 2NF as all non-prime attributes are fully functionally dependent on the primary key of R8.

Relations after Normalizing to Second Normal Form (2NF):
R1 - {user_id, user_name, user_email, user_password, user_sex, user_phone, user_dob, user_created_at, user_is_admin}
R2 - {ticket_id, user_id, payment_id, seat_id, food_preference, created_at, status}
R3 - {payment_id, status, price, created_at}
R4 - {seat_id, train_id, type, price}
R5 - {train_id, train_name, source, destination, start_time, end_time, status, total_seats}
R6 - {station_id, station_name, city}
R7 - {user_id, passenger_id, passenger_name, passenger_email, passenger_sex, passenger_phone, passenger_dob}
R8 - {train_id, station_id, platform, arrival_time, departure_time}

Normalizing to Third Normal Form (3NF):

All the relations stated above are in 2NF and do not contain any transitive dependencies for non-prime attributes hence we can say that they are in the Third Normal Form (3NF).

Relations after Normalizing to Second Normal Form (2NF):
R1 - {user_id, user_name, user_email, user_password, user_sex, user_phone, user_dob, user_created_at, user_is_admin}
R2 - {ticket_id, user_id, payment_id, seat_id, food_preference, created_at, status}
R3 - {payment_id, status, price, created_at}
R4 - {seat_id, train_id, type, price}
R5 - {train_id, train_name, source, destination, start_time, end_time, status, total_seats}
R6 - {station_id, station_name, city}
R7 - {user_id, passenger_id, passenger_name, passenger_email, passenger_sex, passenger_phone, passenger_dob}
R8 - {train_id, station_id, platform, arrival_time, departure_time}

Normalizing to Boyce-Codd Normal Form (BCNF):

For a relation to be in BCNF it is required to be in the 3rd Normal Form, and X should be a superkey for every functional dependency (FD) X−>Y in a given relation.

To check if a relation schema R is in BCNF, it suffices to check only the dependencies
in the given set F for violation of BCNF, rather than checking all dependencies in F+.

Since all relations until now are in **3NF** we will check if they are in **BCNF** with respect to the functional dependencies.

R1 - {user_id, user_name, user_email, user_password, user_sex, user_phone, user_dob, user_created_at, user_is_admin}

Here, user_id determines all the non-prime attributes in its relation, essentially, user_id is the superkey of this relation. Hence R1 is in BCNF

R2 - {ticket_id, user_id, payment_id, seat_id, food_preference, created_at, status}

Here, ticket_id determines all the non-prime attributes in its relation, essentially, ticket_id is the superkey of this relation. Hence R2 is in BCNF

R3 - {payment_id, status, price, created_at}

Here, payment_id determines all the non-prime attributes in its relation, essentially, payment_id is the superkey of this relation. Hence R3 is in BCNF

R4 - {seat_id, train_id, type, price}

Here, seat_id determines all the non-prime attributes in its relation, essentially, seat_id is the superkey of this relation. Hence R4 is in BCNF

R5 - {train_id, train_name, source, destination, start_time, end_time, status, total_seats}

Here, train_id determines all the non-prime attributes in its relation, essentially, seat_id is the superkey of this relation. Hence R5 is in BCNF

R6 - {station_id, station_name, city}

Here, station_id determines all the non-prime attributes in its relation, essentially, station_id is the superkey of this relation. Hence R6 is in BCNF

R7 - {user_id, passenger_id, passenger_name, passenger_email, passenger_sex, passenger_phone, passenger_dob}

Here, user_id and passenger_id determines all the non-prime attributes in its relation, essentially, (user_id,passenger_id) is the superkey of this relation. Hence R7 is in BCNF

R8 - {train_id, station_id, platform, arrival_time, departure_time}

Here, train_id and station_id determines all the non-prime attributes in its relation, essentially, (train_id,station_id) is the superkey of this relation. Hence R7 is in BCNF
All relations R1–R8 are in BCNF and we can now finalize the schemas based on this. Also note that schema that we derived at after using **Reduction** and after **Normalization** are exactly the same which **verifies our database design**.

Finalized Tables:

1. **User**(user_id, user_name, user_email, user_password, user_sex, user_phone, user_dob, user_created_at, user_is_admin)
2. **Ticket**(ticket_id, user_id *[FK],* payment_id *[FK]*, seat_id *[FK]*, food_preference, created_at, status)
3. **Payment**(payment_id, status, price, created_at)
4. **Seat**(seat_id, train_id *[FK]*, type, price)
5. **Train**(train_id, train_name, source, destination, start_time, end_time, status, total_seats)
6. **Station**(station_id, station_name, city)
7. **Passenger**(user_id *[FK]*, passenger_id, passenger_name, passenger_email, passenger_sex, passenger_phone, passenger_dob)
8. **Schedule**(train_id *[FK]*, station_id *[FK]*, platform, arrival_time, departure_time)