Database Management System

Hotel Management Database System - Bernstein's Algorithm and BCNF

Waleed G , Asil K, Kamil K , Arsalan K

Date: 03/19/2023

Creating Transitive and Partial Dependencies:

Table 1: Booking (adding data in the modified tables)

Booking ID	Guest ID	Room Number	Guest Name	Check-in Date	Check-out Date	service_description	amount_paid
1	101	101	John Doe	2023-04-01	2023-04-05	Extra Blankets	$400
2	102	102	Jane Smith	2023-04-02	2023-04-06	Extra Water	$480
3	103	101	Bob Johnson	2023-04-01	2023-04-04	Extra Shampoo	$300

Here are the functional dependencies of the Booking table:

Booking ID, room_id -> Guest ID, Check-in Date, Check-out Date, amount_paid, guest_name, service_description.

guest ID -> guest Name (This is an example of a transitive dependency, as guest_id can determine guest_name although they are both non-key attributes)

room_id-> service_description (This is an example of a partial dependency, as room_id can determine services, as room_id is a part of the compound key, making this a partial dependency as it doesnt need booking_id to determine services)

Displaying All FD's

Hotel: {hotel_id} -> {hotel_name, location}
HotelContact: {hotel_id} -> {hotel_phone_number}
RoomType: {roomtype_id} -> {room_type}
RoomPrice: {roomtype_id, hotel_id} -> {room_price}
Room: {room_id} -> {hotel_id, roomtype_id, requests}
Guest: {guest_id} -> {guest_name, guest_email, guest_phone_number}

Booking:
{Booking ID, room_id} -> {Guest ID, Check-in Date, Check-out Date, amount_paid, guest_name, service_description}
[{guest ID} -> {guest Name}
{room_id} -> {service_description}

Department: {dept_ID} -> {dept_name}
Employee: {employee_id} -> {dept_id, employee_name, job_title, contact, hotel_id}
Service_Type: {service_type_id} -> {service_description}

Algorithms

To normalize the Booking table, we can use Bernstein's algorithm to decompose it into smaller tables that are in 3NF.

First, we list all the functional dependencies:

booking_id, room_id -> guest_id, checkin_date, checkout_date, amount_paid, guest_name, service_description
guest_id -> guest_name
room_id -> service_description
Next, we use these functional dependencies to create a set of tables in 3NF:

Table 2: Guest

guest_id (primary key)
Guest_name

Table 3: Room

room_id (primary key)
hotel_id
roomtype_id
Requests
Service_description

Table 4: Booking

booking_id (primary key)
guest_id (foreign key references Guest(guest_id))
room_id (foreign key references Room(room_id))
checkin_date
checkout_date
amount_paid
The first two tables were created based on the transitive and partial dependencies respectively, and the third table was created to capture the remaining attributes of the original Booking table.

This set of tables is now in 3NF since there are no partial or transitive dependencies in any of the tables.

Steps:

1. List all the functional dependencies for the original table.
2. Use the functional dependencies to create a set of tables in 3NF that capture all the attributes of the original table.
3. Verify that there are no partial or transitive dependencies in any of the new tables.
4. Add foreign key constraints to maintain referential integrity between the tables.

3NF verification:

Here's an explanation of how each functional dependency in the set is in 3NF or BCNF:

Hotel: {hotel_id} -> {hotel_name, location}
This functional dependency is already in 3NF as each non-key attribute (hotel_name, location) is dependent only on the primary key (hotel_id).

HotelContact: {hotel_id} -> {hotel_phone_number}
This functional dependency is already in 3NF as each non-key attribute (hotel_phone_number) is dependent only on the primary key (hotel_id).

RoomType: {roomtype_id} -> {room_type}
This functional dependency is already in 3NF as each non-key attribute (room_type) is dependent only on the primary key (roomtype_id).

RoomPrice: {roomtype_id, hotel_id} -> {room_price}
This functional dependency is already in 3NF as each non-key attribute (room_price) is dependent only on the combination of primary keys (roomtype_id, hotel_id).

Room: {room_id} -> {hotel_id, roomtype_id, requests}
This functional dependency is already in 3NF as each non-key attribute (hotel_id, roomtype_id, requests) is dependent only on the primary key (room_id).

Guest: {guest_id} -> {guest_name, guest_email, guest_phone_number}
This functional dependency is already in 3NF as each non-key attribute (guest_name, guest_email, guest_phone_number) is dependent only on the primary key (guest_id).

Booking: {booking_id} -> {guest_id, room_id, checkin_date, checkout_date, amount_paid}
This functional dependency is already in 3NF as each non-key attribute (guest_id, room_id, checkin_date, checkout_date, amount_paid) is dependent only on the primary key (booking_id).

Department: {dept_ID} -> {dept_name}
This functional dependency is already in 3NF as each non-key attribute (dept_name) is dependent only on the primary key (dept_ID).

Employee: {employee_id} -> {dept_id, employee_name, job_title, contact, hotel_id}
This functional dependency is already in 3NF as each non-key attribute (dept_id, employee_name, job_title, contact, hotel_id) is dependent only on the primary key (employee_id).

Service_Type: {service_type_id} -> {service_description}
This functional dependency is already in 3NF as each non-key attribute (service_description) is dependent only on the primary key (service_type_id).