67-262 Project Phase-2: Airbnb

Team am03: Ron Chew, Mahima Shanware, Kieran Walsh

8 December 2019

Table of Contents

Part 0. Updated Conceptual Model / User Stories

Please find updated Conceptual Model in a separate PDF file.

ID	Type	Role	Goal	Reason
1 RC	Simple 1	A user interested in airbnb	I want to be able to sign up for Airbnb	So that I can start to use its services as a guest or host
2 RC	Simple 2	A guest who is a parent	I want to find a property with a washer and kitchen	So that I can keep my baby clean and fed during my trip
3 RC	Complex 1	A guest	I want to make a booking and pay for my stay (not experience)	So that I can finally go for that dream trip!
4 RC	Complex 2 (NEW)	A guest	Select add-ons for my stay	So I can have additional amenities like cooked meals during my stay
5 MS	Simple 3	A guest	I want to be able to leave a review about the booking	So that I can warn others if I have a bad experience
6 KW	Simple 4	A new potential host	I want to list my property on Airbnb	So that I can host and earn money from it
7 KW	Simple 5	An organized host of a property	I want to set specific dates when my property is available	So I can use the property for personal uses on occasion
8 MS	Analytical 1	A budget conscious host of a property	I want to see how much I will earn this month from the current bookings	So I can plan out my finances with my earnings
9 RC	Simple 6	A potential guest of a property	I want to send a message to the host	So that I can ask questions about the property

10 KW	Complex 3	Customer service representative	I want to identify all sub 2 star reviews from guests and send them a message to contact support	So that I can identify potential customer issues and start addressing them
11 MS	Complex 4	Customer service representative	I want to be able to refund a user	So that I can respond to an unhappy customer
12 MS	Analytical 2	Potential guest	I want to find the average price of a stay by neighborhood.	So that I can find a place that fits my budget.
13 KW	Analytical 3	Customer service representative	I want to find the number of reservations for stays/experiences in a specific city	So that I can recommend the top cities to visit

Part 1. Relational Model

User(**username**, first_name, last_name, email, password, date_of_birth, phone_number, address)
Message(**message_id**, sent_to, sent_by, message_text, datetime_sent)
Host(**username**, verified)
Guest(**username**, banned)

HostReview(**host_review_id,** confirmation_id, checkin_score, location_score, value_score, communication_score, accuracy_score, cleanliness_score, review_text, review_datetime)
GuestReview(**guest_review_id**, confirmation_id, guest_score, review_text, review_datetime)

PaymentOption(**paymentoption_id**, billing_name, billing_address, billing_postalcode, username, paymentoption_type)
PaymentOptionType(**paymentoption_type**)
Payment(**invoice_number**, paymentoption_id, confirmation_id, amount, transaction_datetime)
Payout(**invoice_number**, paymentoption_id, confirmation_id, amount, transaction_datetime)

BankAccount(**paymentoption_id**, account_type, routing_number, account_number)
CreditCard(**paymentoption_id**, card_type, name_on_card, card_number, card_expiration, card_cvv)

Reservation(**confirmation_id**, confirmation_datetime, start_datetime, end_datetime, bookable_id, guest_username)
ReservationAddOns(**confirmation_id, addon_id**)
Addon(**addon_id**, name, price, bookable_id)

Bookable(**bookable_id**, city, host_username, bookable_type)
BookableType(**bookable_type**)

Experience(**experience_bookable_id**, name, duration, description, experience_category_id)
ExperienceCategory(**expererience_category_id**, name)
ExperienceDatePrice(**experience_bookable_id**, **date**, price)

Stay(**stay_bookable_id**, name, stay_category_id, guest_num, bedrooms_num, beds_num, neighborhood, description)
StayDatePrice(**stay_id**, **date**, price_per_night)
StayCategory(**stay_category_id**, name)
Amenity(**amenity_id**, name)
StayHasAmenity(**stay_bookable_id, amenity_id**)

Part 2. Functional Dependencies

User(username -> first_name, last_name, email, password, date_of_birth. phone_number, address)
Message (message_id -> sent_to, sent_by, message_text, datetime_sent)
Host(username -> verified)
Guest(username -> banned)

HostReview(host_review_id -> checkin_score, location_score, value_score, communication_score, accuracy_score, cleanliness_score, review_text, review_datetime)
GuestReview(guest_review_id -> guest_score, review_text, review_datetime)

PaymentOption(paymentoption_id -> billing_name, billing_address, billing_postalcode)
PaymentOptionType(paymentoption_type ->)
Payment(invoice_number -> amount, transaction_datetime)
Payout(invoice_number -> amount, transaction_datetime)

BankAccount(paymentoption_id -> account_type, routing_number, account_number)
CreditCard(paymentoption_id -> card_type, name_on_card, card_number, card_expiration, card_cvv)

Reservation(confirmation_id -> confirmation_datetime, start_datetime, end_datetime)
ReservationAddOns(confirmation_id, addon_id ->)
Addon(addon_id -> name, price)

Bookable(bookable_id -> city)
BookableType(bookable_type ->)

Experience(experience_bookable_id -> name, duration, description)
ExperienceCategory(expererience_category_id -> name)
ExperienceDatePrice(experience_id, date -> price)

Stay(stay_bookable_id -> name, guest_num, bedrooms_num, beds_num, neighborhood, description)
StayDatePrice(stay_id, date -> price_per_night)
StayCategory(stay_category_id -> name)
Amenity(amenity_id -> name)
StayHasAmenity(stay_bookable_id, amenity_id ->)

Part 3. Normalization
We analyze each relation separately:

User (username -> first_name, last_name, email, password, date_of_birth. phone_number, address)
This relation is a good functional dependency because $\{username\}^+ = \{first\_name, last\_name, email, password, date\_of\_birth. phone\_number, address\}$. In addition, there are no partial or transitive dependencies. As the primary key, username determines all of the user's personal information.

Message (message_id -> sent_to, sent_by, message_text, datetime_sent)
$\{message\}^+ = \{sent\_to, sent\_by, message\_text, datetime\_sent\}$, so message determines all of the other attributes of each message. Because there is only one primary key, there are no partial dependencies in this relation. None of the non-primary attributes determine other attributes, so there are no transitive dependencies. For these three reasons, this relation is a good functional dependency.

Host(username -> verified)
Host is a good functional dependency because $\{username\}^+ = \{verified\}$; the primary key determines the non-primary attribute. There is only one primary key and only one non-primary attribute, so there are no partial or transitive dependencies.

Guest(username -> banned)
Guest is a good functional dependency because, like Host, there is only one primary key and one non-primary attribute. By taking the closure of user ($\{user\}^+ = \{banned\}$), we see that user determines banned. Due to the nature of this relation, there are no partial or transitive dependencies.

HostReview(host_review_id -> checkin_score, location_score, value_score, communication_score, accuracy_score, cleanliness_score, review_text, review_datetime)
$\{host\_review\_id\}^+ = \{host\_review\_id, checkin\_score, location\_score, value\_score, communication\_score, accuracy\_score, cleanliness\_score, review\_text, review\_datetime\}$ so the closure of the LHS determines all attributes of the RHS. There will be a specific set of RHS attributes logically for each host_review_id, and because there are no transitive or partial dependencies as well, this is a good functional dependency.

GuestReview(guest_review_id -> guest_score, review_text, review_datetime)
$\{guest\_review\_id\}^+ = \{guest\_review\_id, guest\_score, review\_text, review\_datetime\}$ so the closure of the LHS determines all attributes of the RHS. There are no partial or transitive dependencies. Notice that logically, for each guest_review_id, there should be a specific combination of the RHS attributes, so this is a good functional dependency.

PaymentOption(paymentoption_id -> billing_name, billing_address, billing_postalcode)
When we take the closure of paymentoption_id, the primary key, we get $\{paymentoption\_id\}^+ = \{billing\_name, billing\_address, billing\_postalcode\}$. This shows that paymentoption_id can sufficiently determine all of the other attributes of a user's payment option. There are also no transitive dependencies in this relation. Notice that billing_address does not necessarily determine billing_postalcode; in different cities, there can exist the same street name but different postal codes.

PaymentOptionType(paymentoption_type ->)
$\{paymentoption\_type\}^+ = \{paymentoption\_type\}$, so the LHS determines the RHS. There is only one attribute, so that attribute determines itself. There are no partial or transitive dependencies. This is a good functional dependency.

Payment(invoice_number -> amount, transaction_datetime)
Payment is a good functional dependency because it closure of invoice_number determines all of the other attributes, and because there are no partial or transitive dependencies. There are no partial dependencies because there is only one primary key, and there are no transitive dependencies because no non-primary attributes determine other attributes. Finally, the closure of $\{invoice\_number\}^+ = \{amount, transaction\_datetime\}$, and this shows that the primary key determines all other attributes.

Payout(invoice_number -> amount, transaction_datetime)
Because $\{invoice\_number\}^+ = \{amount, transaction\_datetime\}$, the primary key determines all of the other attributes in the relation. In addition, there are no partial dependencies because there is only one primary key. Lastly, there are no transitive dependencies because none of the non-primary attributes determine other attributes. For these reasons, this is a good functional dependency.

BankAccount(paymentoption_id -> account_type, routing_number, account_number)
We notice here that while the paymentoption_id determines the account_number, the account_number itself then determines routing_number and account_type, meaning it would be in 2NF. Therefore, we split this relation into BankPayment(paymentoption_id -> account_number) and AccountInformation(account_number -> account_type, routing_number). We have now eliminated the transitive dependency, so the two created relations are in good functional dependencies for BCNF.

CreditCard(paymentoption_id -> card_type, name_on_card, card_number, card_expiration, card_cvv)
In CreditCard, there is only one primary key, so there are no partial dependencies. Paymentoption_id determines card_number, but card_number would determine the rest of the attributes, so this would only be 2NF. Because of this, we would split this relation into CreditCard(paymentoption_id -> card_number) and CreditCardInformation(card_number -> card_type, name_on_card, card_expiration, card_cvv).

Reservation(confirmation_id -> confirmation_datetime, start_datetime, end_datetime)
In this relation, there are no partial dependencies because there is only one primary key. There are also no transitive dependencies. In addition, the closure of confirmation_id is $\{confirmation\_id\}^+ = \{confirmation\_datetime, start\_datetime, end\_datetime\}$, which shows that the LHS determines all other attributes on the RHS. Reservation is a good functional dependency.

ReservationAddOns(confirmation_id, addon_id ->)
ReservationAddOns is a good functional dependency because the LHS determines the RHS: $\{confirmation\_id, addon\_id\}^+ = \{confirmation\_id, addon\_id\}$. There are only two attributes in this relation, and both are primary keys, so the closure of the LHS determines all attributes of the relation.

Addon(addon_id -> name, price)
Addon is a good functional dependency because there are no partial and transitive dependencies, and because the closure of addon_id yields $\{addon\_id\}^+ = \{name, price\}$, which means that addon_id can sufficiently determine all of the other attributes of an add-on.

Bookable(bookable_id -> city)
$\{bookable\_id\}^+ = \{bookable\_id, city\}$, so the closure of the LHS determines all attributes of the RHS. Because the city is the only element on the RHS, there are no partial or transitive dependencies. The primary key bookable_id logically only has one city that it will be in, so it is a good functional dependency.

BookableType(bookable_type ->)
This is a good functional dependency as $\{bookable\_type\}^+ = \{bookable\_type\}$. There is only one attribute, so the closure of the LHS determines all attributes of the RHS. There are no partial or transitive dependencies.

Experience(experience_bookable_id -> name, duration, description)
$\{experience\_bookable\_id\}^+ = \{experience\_bookable\_id, name, duration, description\}$ so the closure of the LHS determines all attributes of the RHS. There are no partial or transitive dependencies. Logically, we have that the id will identify all other attributes as there will only be one unique combination for name, duration, and description, so this is a good functional dependency.

ExperienceCategory(expererience_category_id -> name)
$\{expererience\_category\_id\}^+ = \{expererience\_category\_id, name\}$ so the closure of the LHS determines all attributes of the RHS, and because the name is the only element on the RHS, there are no partial or transitive dependencies. The primary key expererience_category_id identifies the name of the category, so it is a good functional dependency.

ExperienceDatePrice(experience_id, date -> price)
$\{experience\_id, date\}^+ = \{experience\_id, date, price\}$, so the closure of the LHS determines all attributes of the RHS, and because the price is the only element on the RHS, there are no partial or transitive dependencies. Logically, the experience_id and the date identify the price of an experience on a certain date. Thus, this is a good functional dependency.

Stay(stay_bookable_id -> name, guest_num, bedrooms_num, beds_num, neighborhood, description)
$\{stay\_bookable\_id\}^+ = \{stay\_bookable\_id, name, guest\_num, bedrooms\_num, beds\_num, neighborhood, description\}$ so the closure of the LHS determines all attributes of the RHS. There are no partial or transitive dependencies. Because the stay_bookable_id can only have one name and one possible combination of the number of guests, which neighborhood, and more, we know that stay_bookable_id is a primary key and thus this is a good functional dependency.

StayDatePrice(stay_id, date -> price_per_night)
$\{stay\_id, date\}^+ = \{stay\_id, date, price\_per\_night\}$ so the closure of the LHS determines all attributes of the RHS and because the price_per_night is the only element on the RHS, there are no partial or transitive dependencies. Logically, the stay_id and the date identify the price of a stay on a certain date, the price_per_night that corresponds. Thus, this is a good functional dependency.

StayCategory(stay_category_id -> name)
$\{stay\_category\}^+ = \{stay\_category, name\}$ so the closure of the LHS determines all attributes of the RHS, and because the name is the only element on the RHS, there are no partial or transitive dependencies. The primary key stay_category identifies the name of the category, so it is a good functional dependency.

Amenity(amenity_id -> name)
$\{amentiy_id\}^+ = \{amenity_id, name\}$so the closure of the LHS determines all attributes of the RHS, and because the name is the only element on the RHS, there are no partial or transitive dependencies. The primary key amenity_id identifies the name of the amenity, so it is a good functional dependency.

StayHasAmenity(stay_bookable_id, amenity_id ->)
This relation is evidently a good functional dependency as $\{stay\_bookable\_id \text{ and } amenity\_id\}^+ = \{stay\_bookable\_id, amenity\_id\}$ so the closure of the LHS determines all attributes of the RHS. Moreover, the two attributes identify themselves. The closure of the LHS determines all attributes of the space StayHasAmenity.

With all the functional dependencies being good, we can say our model is in BCNF.

Part 4. Physical model: Set up the DB

Part 5. Queries