DT139G Normalisering 1NF – 3NF

Kevin Olsson

1. Normalisation till 1NF

Första steget innan man börjar leta efter functional dependencies är att se till att alla respektive relationer redan är normaliserade till 1NF.

Kraven till 1NF

- Alla celler är atomiska (har endast ett värde).
- Det får inte finnas några strukturella repetitioner mellan kolumnerna.
- Alla kolumner innehåller samma datatyper.
- Alla kolumner har unika namn.
- Det finns inga dupliceringar/kopior av några rader.
- Ordningen av rader spelar ingen roll.

Restaurant 1NF

PK	*Restaurant_ID*
	Restaurant_Region_Code
	Restaurant_Region_Name
	Restaurant_Max_Capacity

RestaurantOwner 1NF

PKFK	*Owner_ID*
PKFK	*Restaurant_ID*

Owner 1NF

PK	*Owner_ID*
	OwnerFirstName
	OwnerLastName

Problemet här var att relationen Restaurant hade attributen Owner1_Name, Owner2_name & Owner3_name, vilket skapade en strukturell repetition av kolumner vilket inte få finnas för att en relation ska vara i 1NF. För att lösa detta så skapades en egen relation för Owner, men då relationen mellan Owner till Restaurant är m – n då en ägare kan äga flera restauranger och en restaurang kan ha flera ägare, så behövdes det skapa en mellanrelation ”restaurantOwner” för att skapa en 1-m relation - vilket skapar unika kombinationer av en ägare och en Restaurang.

Relation Dessert 1NF

PK	Dessert_Code
	Dessert_Name
	Dessert_Descr
	Dessert_Cathegory_Code
	Drink_Name
	Topping_Name
	Dessert_Price_Amount

Relation Offer 1NF

PF	Dessert_Code
PF	Restaurant_ID
	Offer_Start_Date
	Offer_End_Date

The attribute ”Dessert_Offered_Date_Range” consisted of multipe values – a start date and also an end date, which goes again the rule of having a relation in 1NF. Additionally this attribute was not needed in the dessert relation to determine the other dessert attributes, and the only context where the date range is interesting to know is in the context of a specific dessert offer – so to address this the attribute has been split into to individual attributes “start” and “end date”, and then these two attributes are inserted into the Offer relation as simple non-key attributes since that is the context where this information is relevant to know. This attribute also caused a partial dependency (since Dessert_Code can determine the other attributes alone) so by removing it from the relation we have already solved that issue for 2NF. (Sorry for suddenly changing to English Language – just noticed.)

Relation Bill 1NF

PK	*Bill_ID*
	Bill_Date_And_Time
	Bill_Amount
	Gratuity_Amount
FK	BoothCode
FK	Dessert_Code
FK	Restaurant_ID

Bill inkluderar inte längre offered_date_range då det inte behövs här och att attributet har förändrat sin struktur. Bortsett från detta så har inga andra förändringar behövt ske för relationen.

Relation Booth 1NF

PK	BoothCode
	Smoke_Section_Indicatior
	Booth_Capacity
	BoothType_Code
	BoothType_Text
FK	Restaurant_ID

Booth är redan i 1NF och har inte behövt göra några förändringar för att uppfylla kraven.

2. Identifiera functional dependencies i relationerna

Relation Restaurant:

PK: Restaurand_ID = A,
RESTAURANT_REGION_CODE = B,
RESTAURANT_REGION_NAME = C,
RESTAURANT_MAX_CAPACITY = D,

Functional dependencies:

A-> B, C, D
B->C

Relation dessert

PK: DESSERT_CODE = A,
DESSERT_NAME = B,
DESSERT_DESCR = C,
DESSERT_CATEGORY_CODE = D,
DRINK_NAME = E,
TOPPNING_NAME = F,
DESSERT_PRICE_AMOUNT = G,

Functional dependencies:

A -> B, C, D, E, F, G

Relation offer

PF: DESSERT_CODE = A,
PF: RESTAURANT_ID = B,
Offer_Start_Date = C,
Offer_End_Date = D,

Functional dependencies:

AB -> C, D

En Dessert_Code tillsammans med en unik restaurang kommer kunna användas för att avgöra datumen för ett offer.

Relation bill

PK: BILL_ID = A,
FK: BOOTHCODE = B,
FK: DESSERT_CODE = C,
FK: RESTAURANT_ID = D,
BILL_DATE_AND_TIME = E,
BILL_AMOUNT = F,
GRATUITY_AMOUNT = G,

Functional dependencies:

A -> B, C, D, E, F, G

Relation booth

PK: BOOTHCODE = A,
FK: RESTAURANT_ID = B,
SMOKE_SECTION_INDICATOR = C,
BOOTH_CAPACITY = D,
BOOTHTYP_CODE = E,
BOOTHTYP_TEXT = F,

Functional dependencies:

A -> BCDEF,
E -> F.

3. Normalisation till 2NF

För att få en relation till 2NF så krävs 1NF + inga partial dependencies – vilket betyder att inga ickenyckel-attribut får vara delvist beroende av kompositа primärnycklar eller kandidatnycklar (Exempel om AB->C men där B->C så är C delvist beroende av B).

Relation restaurant 2NF:

PK	Restaurant_ID
	Restaurant_Region_Code
	Restaurant_Region_Name
	RESTAURANT_MAX_CAPACITY

Inga partial dependencies som förändrar relationen till 2NF (har transitive dependencies via Region code -> Region Name, men de ska åtgärdas för 3NF).

Relation Offer 2NF:

PKFK	Dessert_Code
PKFK	Restaurang_ID
	Offer_Start_Date
	Offer_End_Date

Relation Dessert 2NF:

PK	Dessert_Code
	Dessert_Name
	Dessert_Description
	Drink_Name
	Topping_Name
	Dessert_Price_Amount
FK	Dessert_Cathegory_Code

Det fanns en partial dependency vilket åtgärdades när vi tog bort Offered_Date_Range. Inga andra partial dependencies finns. Enda ändringen är dessert_cathegory_code som nu finns här som en främmande nyckel pga. m-1 relationen som finns mellan dessert och dessert_cathegory.

Relation Dessert_Cathegory 2NF:

PK	Dessert_Cathegory_Code

Så med beskrivningen ovan sagd då blev detta en egen relation med sin egna primärnyckel.

Relation Bill 2NF:

PK	Bill_ID
	Bill_Date_And_Time
	Bill_Amount
	Gratuity_Amount
FK	BoothCode
FK	Dessert_Code
FK	Dessert_Offered_Date_Range
FK	Restaurant_ID

Inga partial dependencies.

Relation Booth 2NF:

PK	BoothCode
	Smoke_Section_Indicatior
	Booth_Capacity
	BoothType_Code
	BoothType_Text
FK	Restaurant_ID

De enda dependencies som hittas här är transitive dependencies –
Sådan att BoothType_Code -> BoothType_Text, vilket ordnas för 3NF.

3NF

2NF + inga transitive dependencies – dvs ett ickenyckel-attribut får inte ha ett funktionellt beroende till ett annat ickenyckel-attribut som i sig har ett beroende till primärnyckeln. Exempel: If A->B & B->C then A->C, vilket inte får finnas.

Hänvisar till ER + UML dokument för att se slutresultatet för 3NF.