CMSC 206: Database Management Systems

Normalisation

Thomas Laurent

26/09/2020

University of the Philippines Open University

Table of contents

Introduction

Introduction

We now have the tools to design DBs, but how to make sure that our DB will be usable, i.e. easy to maintain and query? Until now we have been using our common sense.

The answer: normalisation. Normalisation is the process of transforming a relational model into another, equivalent model that follows a set of rules that make it avoid redundancy and thus make it easier to maintain and query.

The rules of normalisation rely on the concept of functional dependencies that we will introduce this week.

First, let's see an example of a database that is not normalised to get a feeling of why we want to normalise our database

Example of non normalised DB schema[1]

Delivery(SupplierId, SupplierTown, ProductID, ProductType, Quantity)

The supplier *SupplierId* from town *SupplierTown* delivers a *Quantity* of product *ProductID* of type *ProductType*.

Let’s see intuitively why this schema is not optimal.

[1]Example from Professor Jean Yves Martin, Ecole Centrale de Nantes

Example of non normalised DB schema- Problems

As the schema introduces redundancy (SupplierTown and ProductType are recorded for each delivery), it increases the risk of incoherence and problems.

What happens if a supplier changes town and we insert a new tuple with the new town but without updating the database first?

What happens if we do not update all tuples in the database when there is a change?

What happens if we want to record a supplier before he makes a delivery?

Example of non normalised DB

SupplierId	SupplierTown	ProductID	ProductType	Quantity
3	Nantes	52	Furniture	12
22	Paris	10	Computer	6
22	Paris	25	Paper	200
3	Nantes	25	Paper	500
3	Angers	10	Laptop	15

Table 1: A non normalised DB with inconsistencies

If we query the database for the town of supplier 3, what should it return?

The type of product 10 changed, we will not get the same statistic on laptop deliveries if we query using the ProductType column or the ProductID column.

Functional dependencies

Definition

A functional dependency is a **constraint** between two sets of attributes X and Y of a database relational schema.

We say that Y is **functionally dependent on** X, and we note $X \rightarrow Y$ when, for every valid tuples t_1 and t_2 for the database (whether they are in the database or not), if $t_1[X] = t_2[X]$ then $t_1[Y] = t_2[Y]$. i.e. if two tuples have the same value for attributes in X then they have the same values for attributes in Y.

$$(X \rightarrow Y) \Leftrightarrow (\forall t1, t2; t_1[X] = t_2[X] \Rightarrow t_1[Y] = t_2[Y])$$

Remarks

Functional dependencies are part of the semantics of the data, and must be defined by the database designer. They are not intrinsic to the DBMS (but can be implemented through constraints such as keys).

Functional dependencies are dependent on the schema of the database, not on a particular state of the database. It is not possible to determine functional relations from the tuples in the database.

Examples

In the previous *example of non normalised DB schema* we could define the following functional dependencies:

- {DeliveryId} $\rightarrow$ {DeliveryId, SupplierId, SupplierTown, ProductId, ProductType, Quantity}, this is called a **trivial** functional dependency, as DeliveryId is on both the left and right side
- {SupplierId} $\rightarrow$ {SupplierTown}
- {DeliveryId} $\rightarrow$ {ProdcutId}
- {ProductId} $\rightarrow$ {ProducType}, ProductType is **transitively dependent** on DeliveryId via ProductId
- {DeliveryId} $\rightarrow$ {ProductType}
- ...

Normal forms

Introduction

Normal forms are a set of rules that a DB schema should follow to ensure it does not facilitate update anomalies, i.e. that the DB is easy to maintain.

They are several normal forms (e.g. 1NF, 2NF, 3NF, BCNF, 4NF, ETNF, ...), each stricter than the previous one. I will introduce normal forms 1-3 and BCNF here.

Although normal forms are the best practice, they are not always followed in the industry, which can make your life harder when dealing with DBs. After this class I hope you will strive to implement databases that are 3NF/BCNF!

Normal form 1

A schema is of normal form 1 (is 1NF) iff:

- every attribute is *atomic* (i.e. no composite attribute)
- every attribute is monovalued (i.e. no multivalued attributes such as lists)

NB: Considering the definition of a relation, all our schemas should be 1NF

Normal form 2

A schema is of normal form 2 iff:

- it is 1NF
- every attribute not in the primary key depends on the full primary key

Normal form 2 - Example

The previous *example of non normalised DB schema* is not 2NF because of functional dependencies {SupplierId} $\rightarrow$ {SupplierTown} and {ProductId} $\rightarrow$ {ProducType}.

An equivalent schema that is 2NF would be:
Delivery($\underline{\text{SupplierId}}$, $\underline{\text{ProductID}}$, Quantity)
  SupplierId $\rightarrow$ Supplier, ProductId $\rightarrow$ Product

Supplier($\underline{\text{SupplierId}}$, SupplierTown)

Product($\underline{\text{ProductID}}$, ProductType)

Normal form 3

A schema is of normal form 3 iff:

- it is 2NF
- every attribute that does not belong to a candidate key is not transitively dependant on the primary key

N.B: any schema can be made to be 3NF without losing data or functional dependencies. Every DB schema should be 3NF! 3NF still allows redundancy though, prefer BCNF

Normal form 3 - Example

Supplier(<u>SupplierId</u>, SupplierStreet, SupplierTown, SupplierCountry)

This schema is not 3NF as SupplierCountry (not part of the only candidate key {SupplierId}) is transitively dependant on SupplierId (the primary key) via SupplierTown.

An equivalent schema that is 3NF would be:
Supplier(<u>SupplierId</u>, SupplierStreet, SupplierTown)
    SupplierTown $\rightarrow$ Town

Town(<u>TownName</u>, TownCountry)

Boys-Codd Normal form

A schema is in the Boys-Codd normal form iff:

- it is 3NF
- whenever a nontrivial functional dependency $X \rightarrow A$ holds, then X is a superkey

Normals form 4 and 5

These normal forms are less used. For those interested, please see Additional Materials.

Normalisation and table decomposition

Introduction

Now you know what normal forms are, and hopefully the examples have given you an idea of how to transform schemas in order to make them respect the normal forms.

Let us just introduce the concept of decomposition of a relation without loss of information to formalise this intuition a bit.

Decomposition of a relation without loss of information

Let $R(A_1, A_2, ..., A_n)$ a relation.

Let $R_1 = \pi[A_1, A_2, ..., A_i]R$[2], $R_2 = \pi[A_i, ..., A_j]R$, ... $R_k = \pi[A_l, ..., A_n]R$

If $R_1 *$[3]$R_2 * ... * R_k = R$, then we say this is a decomposition of R without loss of information.

Queries on R and on its decomposition would then give the same result.

[2] Projection operator. Formal definition next week. Basically R with only columns $A_1, ..., A_i$

[3] Join operator. Formal definition next week. Basically R_1 and R_2 glued back together

Heath Theorem

Relation $R(X, Y, Z)$ can be decomposed without loss of information into:
$R_1 = \pi[X, Y]$
$R_2 = \pi[X, Z]$

if the functional dependency $X \rightarrow Y$ exists.

Assignment 1

Assignment 1

I have uploaded the instructions for assignment 1, which is about what we saw this week and last week.

The deadline is October 10th, 23:59.

Submit a single PDF report. For exercise 1, each task should have its own diagram, I will not accept a ”final” diagram where I have to guess what part answers which task.

If you make any assumptions (e.g. assuming that X can be linked to multiple Y or that Z is unique), that are not stated in the instructions, make them clear in your report. I can not read your minds (yet ;)). e.g. ”Here I assume that an employee can work for multiple companies at the same time”.