

CPSC301 Tutorial 3 – Normalization

October 3, 2005

Abstract

Now it's time that we make a careful consideration on our schemas derived from the E-R diagram. What are candidate keys for each relation? Are we choosing the correct primary keys? What normal form does each relation belong to? How could we decompose relations so that after a series of actions all relations are in 3NF or BCNF, while the semantics are precisely preserved?

1 Q1

Recall the database schema similar to what we defined in the last tutorial by applying the direct translation rules to the ER diagram:

Schema A (Combining License with Owns and issues):

1. branch(branch_id, branch_name, branch_addr, branch_city, branch_postalcode, branch_phone)
2. license(license_no, license_expiry, license_class, license_type, branch_id, issue_date, driver_sin)
3. driver(driver_sin, driver_name, driver_addr, driver_city, driver_birthdate, driver_postalcode, driver_phone)
4. takes_exam(driver_sin, branch_id, exam_date, type_id, exam_score)

Tasks:

- 1) Find out Functional Dependencies in each relation.
- 2) Examine each relation schema, determine which normal forms they belong to. And (lossless-join + dependency preserving) decompose relations so that after decomposition, all relations are in 3NF.
- 3) Further (lossless-join) decompose relations into BCNF
- 4) Discuss if it's necessary to decompose relations to BCNF. (Hint: From aspect of join burden)

2 A1

The first step of normalization is to list the functional dependencies. There are three sources of functional dependencies:

- * Candidate keys of relations. Each candidate key determines all the other attributes. These keys have been identified from statements in the problem description when the ER diagram was created.

- * One-to-one and many-to-one associations.
 - o If there is a one-to-one association between entity sets E1 and E2, with primary keys A and B respectively then the FD's $A \rightarrow B$ and $B \rightarrow A$ hold.
 - o If there is a many-to-one association from entity sets E1 to E2, whose primary keys are A and B respectively, then the FD $A \rightarrow B$ holds.

- * Other properties of the problem domain which are described in the problem description and cannot be shown in the ER diagram.

To determine whether or not $X \rightarrow Y$ is a functional dependency, ask yourself this question: If two records have the same set of attributes X, can you be certain that they have the same set of attributes Y? If the answer is 'yes', then $X \rightarrow Y$ is a functional dependency. Alternatively, you could ask: If two records do not have the same set of attributes Y, can you be certain that they don't have the same set of attributes X. Again, if the answer is 'yes', then $X \rightarrow Y$ is a functional dependency. For example, $driver_postalcode \rightarrow driver_city$ is a functional dependency because the answer to both questions is 'yes'. However, $driver_postalcode \rightarrow driver_address$ is not a functional dependency because it is possible that two drivers with different addresses share the same postal code (e.g., they live in the same building).

So. In this set of schema, "some" of the FDs can be listed.

license relation

$license_no \rightarrow license_expiry, license_class, license_type$

$license_no \rightarrow driver_sin$

$license_no \rightarrow branch_id, issue_date$

branch relation Candidate keys: $branch_id$, $(branch_addr, branch_city)$

$branch_id \rightarrow branch_name, branch_addr, branch_city, branch_postalcode, branch_phone$

$branch_addr, branch_city \rightarrow branch_name, branch_id, branch_postalcode,$

$branch_phone$

$branch_postalcode \rightarrow branch_city$

driver relation

$driver_sin \rightarrow driver_name, driver_addr, driver_city, driver_birthdate, driver_postalcode,$
 $driver_phone$

$driver_addr, driver_city \rightarrow driver_postalcode,$

$driver_postalcode \rightarrow driver_city$

takes_exam relation

$driver_sin, branch_id, exam_date \rightarrow type_id, exam_score$

[discussion] What other FDs you discover, can they be derived by the listed ones using Armstrong Axioms?

Only branch and driver relations are not in BCNF. (Why?)

branch is in 3rd normal form because

1. $branch_id$ is a key. 2. $branch_addr + branch_city$ is a key 3. $branch_city$ is part of a key.

It is not in BCNF because 3. $branch_city$ is not a key.

Decompose 1st. step to

branch(branch_id, branch_name, branch_addr, branch_city, branch_phone)

address_branch(branch_addr, branch_city, branch_postalcode)

then further decompose address_branch to

address_branch(branch_addr, branch_postalcode)

postalcode_city(branch_postalcode, branch_city)

Now it's in BCNF.

driver relation

Driver not in 3NF because driver_city is not (part of) a key and driver_postalcode is not a super key.

Decompose it to 3NF first.

1. do minimal cover —already is by writing 1. into single terms on the right.

2. decompose the relation into

driver(driver_sin, driver_name, driver_birthdate, driver_phone, driver_addr, driver_city)

addr_driver(driver_addr, driver_city, driver_postalcode,)

Check to see that this is already in 3rd NF.(because we let driver_addr + driver_city to be the key).

For BCNF , we need to further decompose addr_driver into

addr_driver(driver_addr, driver_postalcode)

Driver_postalcode(driver_postalcode, driver_city);

Check to see if it's in BCNF.(very similar to that in branch).

Lastly, For a BCNF design, observe that Driver_postalcode and Branch_postalcode actually model the same semantic. We can combine them into one and change the corresponding references in both relation-sets.

Testing of dependency-preserving is not included in the tutorial. The algorithm is here.[*database system concepts 5th ed.*]

To check if a dependency $\alpha \mapsto \beta$ is preserved in a decomposition of R into R_1, R_2, \dots, R_n we apply the following test (with attribute closure done w.r.t. to F)

result = α

while (changes to result) **do**

for each R_i in the decomposition

t = (result $\cap R_i$)⁺ $\cap R_i$;

result = result \cup t;

endfor

endwhile

If result contains all attributes in β , then the functional dependency $\alpha \mapsto \beta$ is preserved.

We apply the test on all dependencies in F to check if a decomposition is dependency preserving.

This procedure takes polynomial time, instead of the exponential time required to compute F^+ and $(F_1 \cup F_2 \cup \dots \cup F_n)^+$

3 Q2

[RG 3rd P640. 19.9] Case Study: The Internet shop The case study is sketched as follows.

An online book store has come to their db design with E-R diagram as in figure 1.

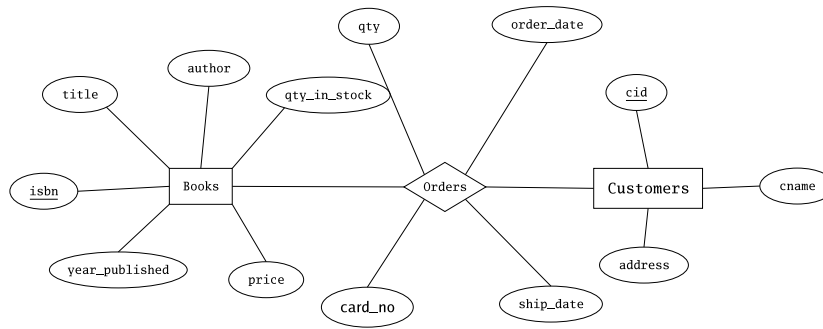


Figure 1: Initial E-R diagram

And a possible relational schema like this.

Books(isbn, title, author, qty_in_stock, price, year_published)

Customer(cid, cname, address)

Orders(ordernum, isbn, cid, cardnum, qty, order_date, ship_date)

Task:

1) Examine this schema, find out functional dependencies and decompose relations into BCNF.

2) Draw the E-R diagram reflecting the final design.

4 A2

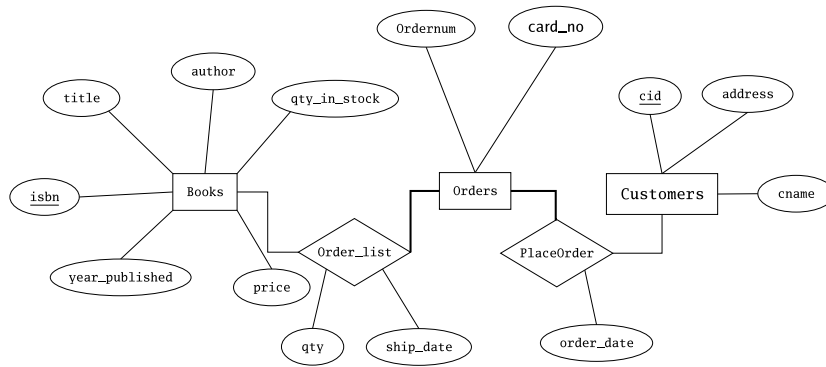
Please refer to textbook.

Decompose orders into BCNF

Orders(ordernum,cid,order_date,cardnum)

Orderlists(ordernum, isbn, qty, ship_date)

And the final E-R design is



5 Comment

Question:

Consider relation $R(J,K,L,M,N)$ with functional dependencies $JK \twoheadrightarrow L$, $KL \twoheadrightarrow M$, $LM \twoheadrightarrow N$, $MN \twoheadrightarrow J$, and $NJ \twoheadrightarrow K$. Project these FD's onto the relation $S(J,K,L,M)$. Which of the following FD's holds in the projected relation?

Answer:

Think of J, K, L, M, N as arranged in a circle. If we start with any two adjacent attributes, such as KL or NJ, then the closure includes all the attributes. However, if we start with any one attribute or any two nonadjacent attributes, then the closure is just what we start with. Notice also that any three or more attributes must include two adjacent attributes, so its closure is all the attributes. To see what FD's hold in S, we must close every subset of J,K,L,M (but allowing N in the closure) and see which among those four attributes are in the closure. In our problem, JK, KL, LM, and any superset has JKLMN as the closure, and thus we get, for S, $JK \twoheadrightarrow LM$, $KL \twoheadrightarrow JM$, and $LM \twoheadrightarrow JK$. We also get FD's that follow from these, such as $JKL \twoheadrightarrow M$. However, S has no nontrivial FD's with singletons on the left or with other pairs on the left.