Database Normalization-Comp.

Contents

4. Boyce Codd Normal Form (BCNF)

In the first part of chapter three, we demonstrated how 2NF and 3NF disallow partial and transitive dependencies on the *primary key* of a relation, respectively. Relations that have these types of dependencies may suffer from the update anomalies.

However, the definition of 2NF and 3NF, respectively, do not consider whether such dependencies remain on other candidate keys of a relation, if any exist. Later we presented general definitions for 2NF and 3NF that disallow partial and transitive dependencies on any *candidate key* of a relation, respectively.

Application of the general definitions of 2NF and 3NF may identify additional redundancy caused by dependencies that violate one or more candidate keys. However, despite these additional constraints, dependencies can still exist that will cause redundancy to be present in 3NF relations. This weakness in 3NF, resulted in the presentation of a stronger normal form called Boyce–Codd Normal Form (Codd, 1974).

BCNF A relation is in BCNF, if and only if, every determinant is a candidate key.

To test whether a relation is in BCNF, we identify all the determinants and make sure that they are candidate keys. Recall that a determinant is an attribute, or a group of attributes, on which some other attribute is fully functionally dependent.

The difference between 3NF and BCNF is that for a functional dependency **A→B**, 3NF allows this dependency in a relation if **B** is a primary-key attribute and **A** is not a candidate key, whereas BCNF insists that for this dependency to remain in a relation, **A** must be a candidate key. Therefore, Boyce–Codd Normal Form is a stronger form of 3NF, such that every relation in BCNF is also in 3NF. However, a relation in 3NF is not necessarily in BCNF.

Example: Suppose you have the following table:

ClientInterview

clientNo	interviewDate	interviewTime	staffNo	roomNo
CR76	13-May-05	10.30	SG5	G101
CR56	13-May-05	12.00	SG5	G101
CR74	13-May-05	12.00	SG37	G102
CR56	1-Jul-05	10.30	SG5	G102

The **ClientInterview** relation has three candidate keys: (clientNo, interviewDate), (staffNo,interviewDate, interviewTime), and (roomNo, interviewDate, interviewTime). Therefore the **ClientInterview** relation has three composite candidate keys, which overlap by sharing the common attribute interviewDate. We select (clientNo, interviewDate) to act as the primary key for this relation. The **ClientInterview** relation has the following form:

ClientInterview (clientNo, interviewDate, interviewTime, staffNo, roomNo)

fd1	clientNo, interviewDate → interviewTime, staffNo, roomNo	(Primary key)
fd2	staffNo, interviewDate, interviewTime → clientNo	(Candidate key)
fd3	roomNo, interviewDate, interviewTime → staffNo, clientNo	(Candidate key)
fd4	staffNo, interviewDate → roomNo	

However, this relation is not in BCNF (a stronger normal form of 3NF) due to the presence of the (staffNo, interviewDate) determinant, which is not a candidate key for the relation. BCNF requires that all determinants in a relation must be a candidate key for the relation. As a consequence the **ClientInterview** relation may suffer from update anomalies.

For example, to change the room number for staff number SG5 on the 13-May-05 we must update two tuples. If only one tuple is updated with the new room number, this results in an inconsistent state for the database.

To transform the **ClientInterview** relation to BCNF, we must remove the violating functional dependency by creating two new relations called **Interview** and **StaffRoom**, as shown:

Interview (clientNo, interviewDate, interviewTime, staffNo)
StaffRoom (staffNo, interviewDate, roomNo)

Interview

clientNo	interviewDate	interviewTime	staffNo
CR76	13-May-05	10.30	SG5
CR56	13-May-05	12.00	SG5
CR74	13-May-05	12.00	SG37
CR56	1-Jul-05	10.30	SG5

StaffRoom

staffNo	interviewDate	roomNo
SG5	13-May-05	G101
SG37	13-May-05	G102
SG5	1-Jul-05	G102

We can decompose any relation that is not in BCNF into BCNF as illustrated. However, it may not always be desirable to transform a relation into BCNF; for example, if there is a functional dependency that is not preserved when we perform the decomposition (that is, the determinant and the attributes it determines are placed in different relations). In this situation, it is difficult to enforce the functional dependency in the relation, and an important constraint is lost. When this occurs, it may be better to stop at 3NF, which always preserves dependencies.

Note in Example, in creating the two BCNF relations from the original **ClientInterview** relation, we have 'lost' the functional dependency,

roomNo,interviewDate, interviewTime → staffNo, clientNo (represented as fd3),

as the determinant for this dependency is no longer in the same relation. However, we must recognize that if the functional dependency,

staffNo, interviewDate→roomNo (represented as fd4)

is not removed, the **ClientInterview** relation will have data redundancy. The decision as to whether it is better to stop the normalization at 3NF or progress to BCNF is dependent on the amount of redundancy resulting from the presence of fd4 and the significance of the 'loss' of fd3. For example, if it is the case that members of staff conduct only one interview per day, then the presence of fd4 in the **ClientInterview** relation will not cause redundancy and therefore the decomposition of this relation into two BCNF relations is not helpful or necessary. On the other hand, if members of staff conduct numerous interviews per day, then the presence of fd4 in the **ClientInterview** relation will cause redundancy and normalization of this relation to BCNF is recommended. However, we should also consider the significance of losing fd3; in other words, does fd3 convey important information about client interviews that must be represented in one of the resulting relations? The answer to this question will help to determine whether it is better to retain all functional dependencies or remove data redundancy.

Example: convert the following table into BCNF:

StaffPropertyInspection

propertyNo	pAddress	iDate	iTime	comments	staffNo	sName	carReg
PG4	6 Lawrence St, Glasgow	18-Oct-03 22-Apr-04 1-Oct-04	10.00 09.00 12.00	Need to replace crockery In good order Damp rot in bathroom	SG37 SG14 SG14	Ann Beech David Ford David Ford	M231 JGR M533 HDR N721 HFR
PG16	5 Novar Dr, Glasgow	22-Apr-04 24-Oct-04	13.00 14.00	Replace living room carpet Good condition	SG14 SG37	David Ford Ann Beech	M533 HDR N721 HFR

First Normal Form (1NF)

We first transfer sample data held on two property inspection reports into table format with rows and columns. This is referred to as the **StaffPropertyInspection** unnormalized table and is shown in the previous table. We identify the key attribute for this unnormalized table as propertyNo. We identify the repeating group in the unnormalized table as the property inspection and staff details, which repeats for each property. The structure of the repeating group is:

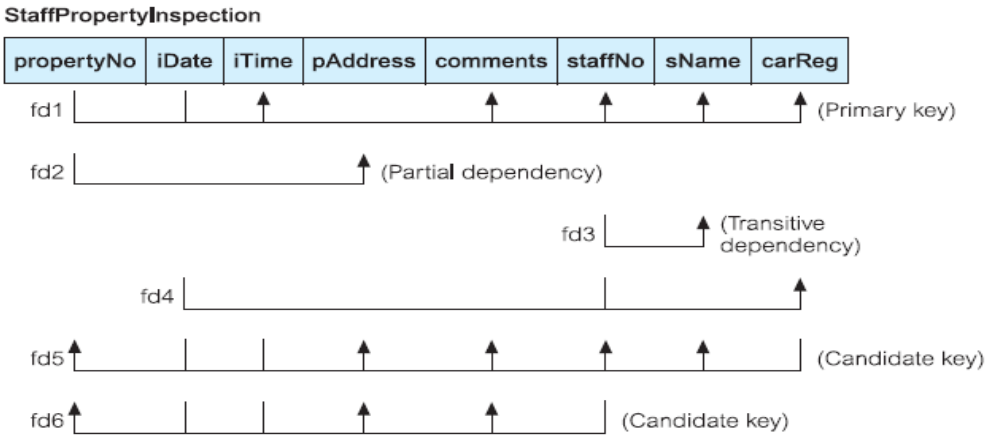
Repeating Group = (iDate, iTime, comments, staffNo, sName, carReg)

StaffPropertyInspection

propertyNo	iDate	iTime	pAddress	comments	staffNo	sName	carReg
PG4	18-Oct-03	10.00	6 Lawrence St, Glasgow	Need to replace crockery	SG37	Ann Beech	M231 JGR
PG4	22-Apr-04	09.00	6 Lawrence St, Glasgow	In good order	SG14	David Ford	M533 HDR
PG4	1-Oct-04	12.00	6 Lawrence St, Glasgow	Damp rot in bathroom	SG14	David Ford	N721 HFR
PG16	22-Apr-04	13.00	5 Novar Dr, Glasgow	Replace living room carpet	SG14	David Ford	M533 HDR
PG16	24-Oct-04	14.00	5 Novar Dr, Glasgow	Good condition	SG37	Ann Beech	N721 HFR

Second Normal Form (2NF):

The normalization of 1NF relations to 2NF involves the removal of partial dependencies on the primary key. If a partial dependency exists, we remove the functionally dependent attributes from the relation by placing them in a new relation with a copy of their determinant.



Using the functional dependencies, we continue the process of normalizing the **StaffPropertyInspection** relation. We begin by testing whether the relation is in 2NF by identifying the presence of any partial dependencies on the primary key. We note that the property attribute (pAddress) is partially dependent on part of the primary key, namely the propertyNo (represented as fd2), whereas the remaining attributes (iTime, comments, staffNo, sName, and carReg) are fully dependent on the whole primary key (propertyNo and iDate), (represented as fd1). Note that although the determinant of the functional dependency

staffNo, iDate → carReg (represented as fd4) only requires the iDate attribute of the primary key, we do not remove this dependency at this stage as the determinant also includes another non-primary-key attribute, namely staffNo. In other words, this dependency is *not* wholly dependent on part of the primary key and therefore does not violate 2NF.

The identification of the partial dependency (propertyNo → pAddress) indicates that the **StaffPropertyInspection** relation is not in 2NF. To transform the relation into 2NF requires the creation of new relations so that the attributes that are not fully dependent on the primary key are associated with only the appropriate part of the key. The **StaffPropertyInspection** relation is transformed into second normal form by removing the partial dependency from the relation and creating two new relations called Property and **PropertyInspection** with the following form:

Property (propertyNo, pAddress)

PropertyInspection (propertyNo, iDate, iTime, comments, staffNo, sName, carReg)

These relations are in 2NF, as every non-primary-key attribute is functionally dependent on the primary key of the relation.

Third Normal Form (3NF):

The normalization of 2NF relations to 3NF involves the removal of transitive dependencies. If a transitive dependency exists, we remove the transitively dependent attributes from the relation by placing them in a new relation along with a copy of their determinant. The functional dependencies within the Property and **PropertyInspection** relations are as follows:

Property Relation

fd2 propertyNo → pAddress

PropertyInspection Relation

fd1 propertyNo, iDate → iTime, comments, staffNo, sName, carReg

fd3 staffNo → sName

fd4 staffNo, iDate → carReg

fd5′ carReg, iDate, iTime → propertyNo, comments, staffNo, sName

fd6′ staffNo, iDate, iTime → propertyNo, comments

As the Property relation does not have transitive dependencies on the primary key, It is therefore already in 3NF. However, although all the non-primary-key attributes within the **PropertyInspection** relation are functionally dependent on the primary key, sName is also transitively dependent on staffNo (represented as fd3). We also note the functional dependency

staffNo, iDate → carReg (represented as fd4) has a non-primary-key attribute carReg partially dependent on a non-primary-key attribute, staffNo. We do not remove this dependency at this stage as part of the determinant for this dependency includes a primarykey attribute, namely iDate. In other words, this dependency is *not* wholly transitively dependent on non-primary-key attributes and therefore does not violate 3NF.

(In other words, when considering all candidate keys of a relation, the staffNo, iDate → carReg dependency is allowed in 3NF because carReg is a primarykey attribute as it is part of the candidate key (carReg, iDate, iTime) of the original PropertyInspection relation.)

To transform the **PropertyInspection** relation into 3NF, we remove the transitive dependency (staffNo→sName) by creating two new relations called **Staff** and **PropertyInspect** with the form:

Staff (staffNo, sName)

PropertyInspect (propertyNo, iDate, iTime, comments, staffNo, carReg)

The Staff and **PropertyInspect** relations are in 3NF as no non-primary-key attribute is wholly functionally dependent on another non-primary-key attribute.

Thus, the **StaffPropertyInspection** relation has been transformed by the process of normalization into three relations in 3NF with the following form:

Property (propertyNo, pAddress)

Staff (staffNo, sName)

PropertyInspect (propertyNo, iDate, iTime, comments, staffNo, carReg)

Boyce–Codd Normal Form (BCNF):

We now examine the Property, Staff, and PropertyInspect relations to determine whether they are in BCNF. Recall that a relation is in BCNF if every determinant of a relation is a candidate key. Therefore, to test for BCNF, we simply identify all the determinants and make sure they are candidate keys.The functional dependencies for the Property, Staff, and PropertyInspect relations are as follows:

Property Relation
fd2 propertyNo → pAddress

Staff Relation
fd3 staffNo → sName

PropertyInspect Relation
fd1′ propertyNo, iDate → iTime, comments, staffNo, carReg
fd4 staffNo, iDate → carReg
fd5′ carReg, iDate, iTime → propertyNo, comments, staffNo
fd6′ staffNo, iDate, iTime → propertyNo, comments

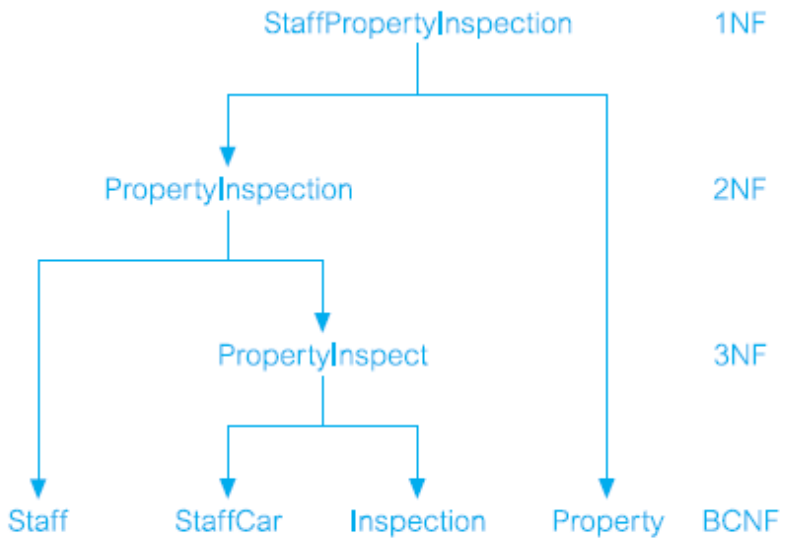
We can see that the **Property** and **Staff** relations are already in BCNF as the determinant in each of these relations is also the candidate key. The only 3NF relation that is not in BCNF is **PropertyInspect** because of the presence of the determinant (staffNo, iDate), which is not a candidate key (represented as fd4). As a consequence the PropertyInspect relation may suffer from update anomalies. For example, to change the car allocated to staff number SG14 on the 22-Apr-03, we must update two tuples. If only one tuple is updated with the new car registration number, this results in an inconsistent state for the database.

To transform the PropertyInspect relation into BCNF, we must remove the dependency that violates BCNF by creating two new relations called StaffCar and Inspection with the form:

StaffCar (staffNo, iDate, carReg)

Inspection (propertyNo, iDate, iTime, comments, staffNo)

The **StaffCar** and **Inspection** relations are in BCNF as the determinant in each of these relations is also a candidate key.



5. Fourth Normal Form (4NF)

Although BCNF removes any anomalies due to functional dependencies, further research led to the identification of another type of dependency called a **Multi-Valued Dependency** (MVD), which can also cause data redundancy (Fagin, 1977). In this section, we briefly describe a multi-valued dependency and the association of this type of dependency with Fourth Normal Form (4NF).

The possible existence of multi-valued dependencies in a relation is due to First Normal Form, which disallows an attribute in a tuple from having a set of values. For example, if we have two multi-valued attributes in a relation, we have to repeat each value of one of the attributes with every value of the other attribute, to ensure that tuples of the relation are consistent. This type of constraint is referred to as a multi-valued dependency and results in data redundancy.

Consider the **BranchStaffOwner** relation shown in Figure, which displays the names of members of staff (sName) and property owners (oName) at each branch office (branchNo). In this example, assume that staff name (sName) uniquely identifies each member of staff and that the owner name (oName) uniquely identifies each owner.

BranchStaffOwner

branchNo	sName	oName
B003	Ann Beech	Carol Farrel
B003	David Ford	Carol Farrel
B003	Ann Beech	Tina Murphy
B003	David Ford	Tina Murphy

Multi-Valued Dependency (MVD) Represents a dependency between attributes (for example, A, B, and C) in a relation, such that for each value of A there is a set of values for B and a set of values for C. However, the set of values for B and C are independent of each other. We represent a MVD between attributes A, B, and C in a relation using the following notation:

A –>> B

A –>> C

For example, we specify the MVD in the **BranchStaffOwner** relation shown in Figure shows:

branchNo –>> sName

branchNo –>> oName

Even though the **BranchStaffOwner** relation is in BCNF, the relation remains poorly structured, due to the data redundancy caused by the presence of the

nontrivial MVD. We clearly require a stronger form of BCNF that prevents relational structures such as the **BranchStaffOwner** relation.

BranchStaff

branchNo	sName
B003	Ann Beech
B003	David Ford

BranchOwner

branchNo	oName
B003	Carol Farrel
B003	Tina Murphy

6. Fifth Normal Form (5NF)

Whenever we decompose a relation into two relations the resulting relations have the lossless-join property. This property refers to the fact that we can rejoin the resulting relations to produce the original relation. However, there are cases were there is the requirement to decompose a relation into more than two relations. Although rare, these cases are managed by join dependency and Fifth Normal Form (5NF).

In this section we briefly describe the lossless-join dependency and the association with 5NF.

Lossless-join dependency A property of decomposition, which ensures that no spurious tuples are generated when relations are reunited through a natural join operation.

In splitting relations by projection, we are very explicit about the method of decomposition. In particular, we are careful to use projections that can be reversed by joining the resulting relations, so that the original relation is reconstructed. Such a decomposition is called a **lossless-join** (also called a *nonloss-* or *nonadditive-* join) decomposition, because it preserves all the data in the original relation and does not result in the creation of additional spurious tuples. For example, Figures

(a) and (b) show that the decomposition of the **BranchStaffOwner** relation into the **BranchStaff** and **BranchOwner** relations has the lossless-join property.

BranchStaff

branchNo	sName
B003	Ann Beech
B003	David Ford

BranchOwner

branchNo	oName
B003	Carol Farrel
B003	Tina Murphy

In other words, the original BranchStaffOwner relation can be reconstructed by performing a natural join operation on the BranchStaff and BranchOwner relations. In this example, the original relation is decomposed into two relations. However, there are cases were we require to perform a lossless-join decompose of a relation into more than two relations (Aho *et al*., 1979). These cases are the focus of the lossless-join dependency and Fifth Normal Form (5NF).

Join dependency Describes a type of dependency. For example, for a relation R with subsets of the attributes of R denoted as A, B, . . . , Z, a relation R satisfies a join dependency if and only if every legal value of R is equal to the join of its projections on A, B, . . . , Z.

Example: suppose you have the following relation:

COURSE
SUBJECT
LECTURER
CLASS

SUBJECT	LECTURER	CLASS
Mathematics	Alex	SEMESTER 1
Mathematics	Rose	SEMESTER 1
Physics	Rose	SEMESTER 1
Physics	Joseph	SEMESTER 2
Chemistry	Adam	SEMESTER 1

In above table, Rose takes both Mathematics and Physics class for Semester 1, but she does not take Physics class for Semester 2. In this case, combination of all these 3 fields is required to identify a valid data. Imagine we want to add a new class - Semester3 but do not know which Subject and who will be taking that subject. We would be simply inserting a new entry with Class as Semester3 and leaving Lecturer and subject as NULL. As we discussed above, it's not a good to have such entries. Moreover, all the three columns together act as a primary key, we cannot leave other two columns blank!

Hence we have to decompose the table in such a way that it satisfies all the rules till 4NF and when join them by using keys, it should yield correct record. Here, we can represent each lecturer's Subject area and their classes in a better way. We can divide above table into three - (SUBJECT, LECTURER), (LECTURER, CLASS), (SUBJECT, CLASS).

5NF

SUBJECT	LECTURER
Mathematics	Alex
Mathematics	Rose
Physics	Rose
Physics	Joseph
Chemistry	Adam

CLASS	LECTURER
SEMESTER 1	Alex
SEMESTER 1	Rose
SEMESTER 1	Rose
SEMESTER 2	Joseph
SEMESTER 1	Adam

CLASS	SUBJECT
SEMESTER 1	Mathematics
SEMESTER 1	Physics
SEMESTER 1	Chemistry
SEMESTER 2	Physics

7. Sixth Normal Form (6NF)

In order for a table to be in 6NF, it has to comply with the 5NF first and then it requires that each table satisfies only trivial join dependencies.

The sixth normal form is currently being used in some data warehouses where the benefits outweigh the drawbacks, for example using Anchor Modeling. Although using 6NF leads to an explosion of tables, modern databases can prune the tables from select queries (using a process called 'table elimination') where they are not required and thus speed up queries that only access several attributes.

Trivial Dependency: If an FD X → Y holds where Y subset of X, then it is called a trivial FD. Trivial FDs are always hold.