Virtual Lab Computer Science and Engineering

IIIT Hyderabad: Database LAB

JUIT Lab: Database LAB (10B17CI372)

Exp-1: Third Normal Form

Introduction

The Third normal form (3NF) is an important form of database normalization. 3NF is said to hold if and only if both of the following conditions hold:

1. The relation *R* (table) is in second normal form (2NF)
2. Every non-prime attribute of *R* is non-transitively dependent (i.e. directly dependent) on every candidate key of *R*.

A non-prime attribute of *R* is an attribute that does not belong to any candidate key of *R*. A transitive dependency is a functional dependency in which $X \rightarrow Z$ (*X* determines *Z*) indirectly, by virtue of $X \rightarrow Y$ and $Y \rightarrow Z$ (where it is not the case that $Y \rightarrow X$).

Theory

There are two basic requirements for a database to be in third normal form:

- Already meet the requirements of both 1NF and 2NF
- Remove columns that are not fully dependent upon the primary key.

Imagine that we have a table of widget orders that contains the following attributes:

- Order Number
- Customer Number
- Unit Price
- Quantity
- Total

Remember, our first requirement is that the table must satisfy the requirements of 1NF and 2NF. Are there any duplicative columns? No. Do we have a primary key? Yes, the order number. Therefore, we satisfy the requirements of 1NF. Are there any subsets of data that apply to multiple rows? No, so we also satisfy the requirements of 2NF.

Now, are all of the columns fully dependent upon the primary key? The customer number varies with the order number and it doesn't appear to depend upon any of the other fields. What about the unit price? This field could be dependent upon the customer number in a situation where we charged each customer a set price. However, looking at the data above, it appears we sometimes charge the same customer different prices. Therefore, the unit price is fully dependent upon the order number. The quantity of items also varies from order to order, so we're OK there.

What about the total? It looks like we might be in trouble here. The total can be derived by multiplying the unit price by the quantity, therefore it's not fully dependent upon the primary key. We must remove it from the table to comply with the third normal form. Perhaps we use the following attributes:

- Order Number
- Customer Number
- Unit Price
- Quantity

Now our table is in 3NF. But, you might ask, what about the total? This is a derived field and it's best not to store it in the database at all. We can simply compute it "on the fly" when performing database queries. For example, we might have previously used this query to retrieve order numbers and totals:

```
SELECT OrderNumber, Total
FROM WidgetOrders
```

We can now use the following query:

```
SELECT OrderNumber, UnitPrice * Quantity AS Total
FROM WidgetOrders
```

to achieve the same results without violating normalization rules.

Procedure

Kindly refer to the common procedure page for details of how to use this platform.

Simulation

Lets consider the following example of a database of a tournament and the details of its winner. We represent the dependencies between the attributes of this table using " <Attribute1> : <Attribute2> ", which means that Attribute1 depends on Attribute2.

```
table = """ Tournament Year Winner Date_of_Birth """ primary_key
= """ Tournament Year """ Dependencies = [ 'Winner :
Date_of_Birth' ]
```

We get the following table after normalizing the table into Third Normal Form.

```
import re def convertTo3NF(): table1 = table.strip() tokens =
re.split("\s+", table1) for i in Dependencies: rel = i.split(" :
") l = [] l.append(rel[1].strip()) print rel[0],rel[1] tokens =
list(set(tokens) - set(l)) for i in tokens: print i,
convertTo3NF()
```

```
    Winner Date_of_Birth
    Tournament Winner Year
```

Quiz

1. Consider the following Relation Schema. EMP_DEPT (ENAME, ENO, DOB, ADDRESS, DNO, DNAME, DMANAGER). Convert it into Third Normal Form.
2. Consider the following Schema. CAR_SALE (Car#, Date_Sold, Salesman#, Commission%, Discount_amt). Convert it into Third Normal Form.
3. Consider the following relation for publised books. BOOK (Book_title, Authorname, Book_type, price, Author_aff, Publisher). Convert it into 3NF.

References

1. http://c2.com/cgi/wiki?ThirdNormalForm
2. Database Management systems by Elmasri and Navathe
3. http://en.wikipedia.org/wiki/Third_normal_form

4. http://databases.about.com/od/administration/l/bldef_3nf.htm
5. http://www.blueclaw-db.com/database_3rd_normal_form.htm

Exp-2: Boyce-Codd Normal Form

Introduction

A relation R is in Boyce-Codd normal form (BCNF) if and only if every determinant is a candidate key. The definition of BCNF addresses certain (rather unlikely) situations which 3NF does not handle. The characteristics of a relation which distinguish 3NF from BCNF are given below. Since it is so unlikely that a relation would have these characteristics, in practical real-life design it is usually the case that relations in 3NF are also in BCNF. Since relations in 3NF but not in BCNF are slightly unusual, it is a bit more difficult to come up with meaningful examples. To be precise, the definition of 3NF does not deal with a relation that:

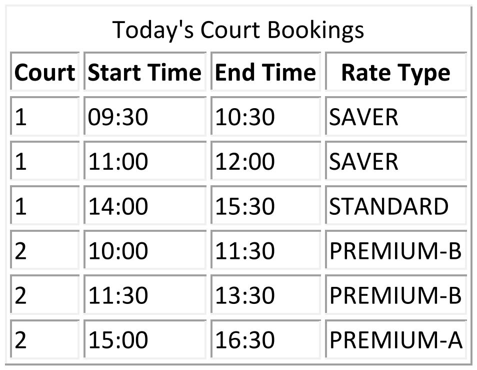
1. has multiple candidate keys, where
2. those candidate keys are composite, and
3. the candidate keys overlap (i.e., have at least one common attribute)

Theory/Procedure

3NF tables not meeting BCNF

Only in rare cases does a 3NF table not meet the requirements of BCNF. A 3NF table which does not have multiple overlapping candidate keys is guaranteed to be in BCNF. Depending on what its functional dependencies are, a 3NF table with two or more overlapping candidate keys may or may not be in BCNF

An example of a 3NF table that does not meet BCNF is:



Today's Court Bookings

Court	Start Time	End Time	Rate Type
1	09:30	10:30	SAVER
1	11:00	12:00	SAVER
1	14:00	15:30	STANDARD
2	10:00	11:30	PREMIUM-B
2	11:30	13:30	PREMIUM-B
2	15:00	16:30	PREMIUM-A

- Each row in the table represents a court booking at a tennis club that has one hard court (Court 1) and one grass court (Court 2)

- A booking is defined by its Court and the period for which the Court is reserved
- Additionally, each booking has a Rate Type associated with it. There are four distinct rate types:
    - SAVER, for Court 1 bookings made by members
    - STANDARD, for Court 1 bookings made by non-members
    - PREMIUM-A, for Court 2 bookings made by members
    - PREMIUM-B, for Court 2 bookings made by non-members

The table's superkeys are:

- S_1 = {Court, Start Time}
- S_2 = {Court, End Time}
- S_3 = {Rate Type, Start Time}
- S_4 = {Rate Type, End Time}
- S_5 = {Court, Start Time, End Time}
- S_6 = {Rate Type, Start Time, End Time}
- S_7 = {Court, Rate Type, Start Time}
- S_8 = {Court, Rate Type, End Time}
- S_T = {Court, Rate Type, Start Time, End Time}, the trivial superkey

Note that even though in the above table *Start Time* and *End Time* attributes have no duplicate values for each of them, we still have to admit that in some other days two different bookings on court 1 and court 2 could *start at the same time* or *end at the same time*. This is the reason why {Start Time} and {End Time} cannot be considered as the table's superkeys.

However, only S_1 to S_4 are candidate keys (that is, minimal superkeys for that relation) because e.g. $S_1 \subset S_5$, so S_5 cannot be a candidate key.

Recall that 2NF prohibits partial functional dependencies of non-prime attributes (ie an attribute that does not occur in ANY candidate key) on candidate keys, and that 3NF prohibits transitive functional dependencies of non-prime attributes on candidate keys.

In **Today's Court Bookings** table, there are no non-prime attributes: that is, all attributes belong to some candidate key. Therefore the table adheres to both 2NF and 3NF.

The table does not adhere to BCNF. This is because of the dependency Rate Type → Court, in which the determining attribute (Rate Type) is neither a candidate key nor a superset of a candidate key.

Dependency Rate Type → Court is respected as a Rate Type should only ever apply to a single Court.

The design can be amended so that it meets BCNF:

Rate Types

Rate Type	**Court**	**Member Flag**
SAVER	1	Yes
STANDARD	1	No
PREMIUM-A	2	Yes
PREMIUM-B	2	No

Today's Bookings

Rate Type	**Start Time**	**End Time**
SAVER	09:30	10:30
SAVER	11:00	12:00
STANDARD	14:00	15:30
PREMIUM-B	10:00	11:30
PREMIUM-B	11:30	13:30
PREMIUM-A	15:00	16:30

Example: The candidate keys for the Rate Types table are {Rate Type} and {Court, Member Flag}; the candidate keys for the Today's Bookings table are {Rate Type, Start Time} and {Rate Type, End Time}. Both tables are in BCNF. Having one Rate Type associated with two different Courts is now impossible, so the anomaly affecting the original table has been eliminated.

Achievability of BCNF

In some cases, a non-BCNF table cannot be decomposed into tables that satisfy BCNF and preserve the dependencies that held in the original table. Beeri and Bernstein showed in 1979 that, for example, a set of functional dependencies {AB → C, C → B} cannot be represented by a BCNF schema. Thus, unlike the first three normal forms, BCNF is not always achievable.

Consider the following non-BCNF table whose functional dependencies follow the {AB → C, C → B} pattern:

Nearest Shops

Person	Shop Type	Nearest Shop
Davidson	Optician	Eagle Eye
Davidson	Hairdresser	Snippets
Wright	Bookshop	Merlin Books
Fuller	Bakery	Doughy's
Fuller	Hairdresser	Sweeney Todd's
Fuller	Optician	Eagle Eye

For each Person / Shop Type combination, the table tells us which shop of this type is geographically nearest to the person's home. We assume for simplicity that a single shop cannot be of more than one type.

The candidate keys of the table are:

- {Person, Shop Type}
- {Person, Nearest Shop}

Because all three attributes are prime attributes (i.e. belong to candidate keys), the table is in 3NF. The table is not in BCNF, however, as the Shop Type attribute is functionally dependent on a non-superkey: Nearest Shop.

The violation of BCNF means that the table is subject to anomalies. For example, Eagle Eye might have its Shop Type changed to "Optometrist" on its "Fuller" record while retaining the Shop Type "Optician" on its "Davidson" record. This would imply contradictory answers to the question: "What is Eagle Eye's Shop Type?" Holding each shop's Shop Type only once would seem preferable, as doing so would prevent such anomalies from occurring:

Shop Near Person

Person	Shop
Davidson	Eagle Eye
Davidson	Snippets
Wright	Merlin Books
Fuller	Doughy's
Fuller	Sweeney Todd's
Fuller	Eagle Eye

Shop

Shop	Shop Type
Eagle Eye	Optician
Snippets	Hairdresser
Merlin Books	Bookshop
Doughy's	Bakery
Sweeney Todd's	Hairdresser

In this revised design , the "Shop Near Person" table has a candidate key of {Person, Shop}, and the "Shop" table has a candidate key of {Shop}. Unfortunately, although this design adheres to BCNF, it is unacceptable on different grounds: it allows us to record multiple shops of the same type against the same person. In other words, its candidate keys do not guarantee that the functional dependency {Person, Shop Type} → {Shop} will be respected.

A design that eliminates all of these anomalies (but does not conform to BCNF) is possible. This design consists of the original "Nearest Shops" table supplemented by the "Shop" table described above.

Nearest Shops

Person	Shop Type	Nearest Shop
Davidson	Optician	Eagle Eye
Davidson	Hairdresser	Snippets
Wright	Bookshop	Merlin Books
Fuller	Bakery	Doughy's
Fuller	Hairdresser	Sweeney Todd's
Fuller	Optician	Eagle Eye

Shop

Shop	Shop Type
Eagle Eye	Optician
Snippets	Hairdresser
Merlin Books	Bookshop
Doughy's	Bakery
Sweeney Todd's	Hairdresser

If a referential integrity constraint is defined to the effect that {Shop Type, Nearest Shop} from the first table must refer to a {Shop Type, Shop} from the second table, then the data anomalies described previously are prevented.

Simulation

Lets consider the following example of a university database consisting of a student, the courses he has taken and the instructor for the course. Similar to the third normal form, we represent the dependencies between the attributes of this table using " : ", which means that Attribute1 depends on Attribute2.

```
table = """ Student Course Instructor """ primary_key = """ Student Course
""" Dependencies = [ 'Instructor : Course' ]
```

After normalizing the above database to be in BCNF, we get the following.

Quiz

1. Prove that any relational schema with two attributes is in BCNF.
2. Convert the given Relation Schema into BCNF. TEACH (Student, Course, Instructor).
3. Consider the following relation for publised books. BOOK (Book_title, Authorname, Book_type, price, Author_aff, Publisher). Convert it into BCNF.

References

1. http://en.wikipedia.org/wiki/Boyce%E2%80%93Codd_normal_form
2. Database Management systems by Elmasri and Navathe
3. http://databases.about.com/cs/specificproducts/g/bcnf.htm
4. http://db.grussell.org/section009.html