
BCNF

Boyce Codd Normal Form (BCNF)

- A relation is in BCNF, if and only if every determinant is a candidate key.
- What is a determinant?
  - Suppose FD X -> Y (e.g., X is CustomerID, Y is CustomerEmail)
  - Y is functionally dependent on X.
  - X is the determinant and Y is the dependent.
- So how do you check if a relation is in BCNF?
  - Look at the LHS (left-hand-side) of the arrow OF EVERY FD, and figure out if it is a candidate key
- Recall that a candidate key is a set of attributes that can be uniquely used to identify a tuple. (it is a minimal superkey)
- Hence, just ask if whatever is there on the LHS of the arrow OF EVERY FD can be used to uniquely identify a tuple in a relation
  - If YES, the relation is in BCNF
  - If NO, the relation is not in BCNF

Normal Forms

- A relation R is in *Boyce-Codd normal form* if for every FD X → A that holds on R, X is a superkey of R.

- Increasing Order of restrictiveness: 1NF, 2NF, 3NF, BCNF. For example, if a relation schema R is in BCNF, it is in 3NF.

Quick example

- Consider a relation R = (A, B, C, D)
  - Suppose FDs are C □ A and {C, D} □ B
  - Suppose candidate keys are CD and AC.

- Does {C, D} □ B satisfy BCNF?
  - Look at the LHS
  - Does {C, D} a candidate key?
    - YES!
  - Hence, it satisfies BCNF

- Does C □ A satisfy BCNF?
  - Ask if the LHS is a candidate key
  - Is C a candidate key?
    - NO!
  - Hence, it violates BCNF
- REMEMBER that for a relation R to be in BCNF, the LHS of every FD must be a candidate key
- In the above example, since the LHS of the FD C □ A is not a candidate key, the relation R is not in BCNF

Learning points

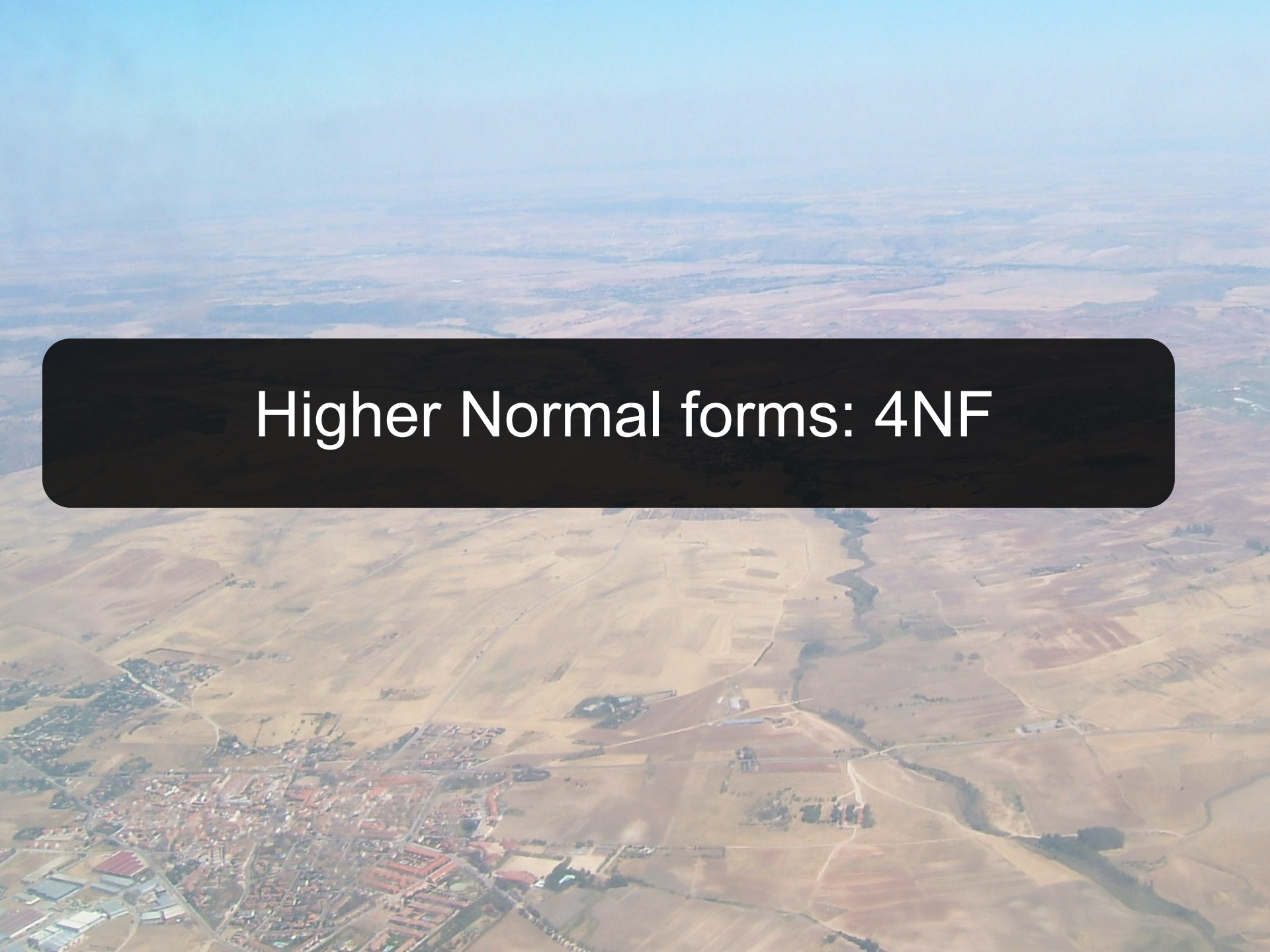
To figure out if a relation R is in BCNF, check if the LHS of every FD is a candidate key

BCNF is stronger than 3NF

If a table is in BCNF, it implies that it is also in 3NF

Conversion from 3NF to BCNF

- Identify all FDs (and also all candidate keys)
- If an FD exists such that its LHS is not a candidate key, create a new table with the LHS as the key. And add the RHS of the FD to that new table.
- In short, it means that you are removing all FDs whose LHS is violating BCNF
  - Sometimes, you may not be able to preserve FDs while decomposing to BCNF


Higher Normal forms: 4NF

4NFand MVDs

- 4NF multivalued dependencies
- What does multivalued dependency (MVD) mean?
  - Recall that FD *A -> B* is about one value to A determining one value of B. (Think of it as one-to-one)
  - MVD *A ->-> B is about one value of A determining multiple values of B. (Think of it as one-to-many)*
  - Example: Consider a relation R (studentID, Hobbies, StudentEmail)
  - Observe that this table is in 3NF (no atomic values, no partial dependencies, no transitive dependencies)
  - ONE value of studentID will determine multiple values of Hobbies.
  - ONE value of studentID will determine multiple values of StudentEmail.

Example

- Suppose studentID S001 has three hobbies such as singing, dancing, reading.
- Suppose his email IDs are abc@gmail.comSuppose his email IDs are abc@gmail.com, xyz@gmail.com
- Is there any dependency between Hobbies and email addresses? NO!
- Now think of different ways in which you can represent this information in a table.

This example (this slide and several subsequent slides) has been inspired by the example provided at http://www.cs.jcu.edu.au/Subjects/cp1500/1998/Lecture_Notes/normalisation/mvd.html

Different ways of putting this info into a table

StudentID	Hobbies	StudentEmail
S001	Singing	abc@gmail.com
S001	Dancing	xyz@gmail.com
S001	Reading	NULL

Different ways of putting this info into a table

StudentID	Hobbies	StudentEmail
S001	Singing	abc@gmail.com
S001	Dancing	xyz@gmail.com
S001	Reading	NULL

Bad design because

1) Hobbies and email are unrelated
2) NULL creates problems

This table still suffers from issues of anomalies

StudentID	Hobbies	StudentEmail
S001	Singing	abc@gmail.com
S001	Dancing	xyz@gmail.com
S001	Reading	abc@gmail.com

Different ways of putting this info into a table

StudentID	Hobbies	StudentEmail
S001	Singing	abc@gmail.com
S001	Dancing	xyz@gmail.com
S001	Reading	NULL

Bad design because

1) Hobbies and email are unrelated
2) NULL creates problems

This table still suffers from issues of anomalies

StudentID	Hobbies	StudentEmail
S001	Singing	abc@gmail.com
S001	Dancing	xyz@gmail.com
S001	Reading	abc@gmail.com

Although you removed NULL, hobbies and email are still unrelated

This table still suffers from issues of anomalies!

Good decomposition

- There can be several ways of putting this information into a table (many different permutations and combinations)
- But all these ways suffer from issues of anomalies
- So what would be a good decomposition?

Good decomposition

StudentID	Hobbies
S001	Singing
S001	Dancing
S001	Reading

StudentID	StudentEmail
S001	abc@gmail.com
S001	xyz@gmail.com

Does this make more sense? YES!

Good decomposition

StudentID	Hobbies
S001	Singing
S001	Dancing
S001	Reading

StudentID	StudentEmail
S001	abc@gmail.com
S001	xyz@gmail.com

Does this make more sense? YES!

This is now in 4NF!

4NF

- A relation is in 4NF if
  - it is in 3NF
  - And no MVDs exist
- Note that both the above conditions must be satisfied for a relation to be in 4NF
- In short, it means that remove all multi-valued dependencies from a table (which is in 3NF) and then that table will be in 4NF.

4NF remove multi-valued dependencies

4NF

- Look back at the example of R(StudentID, Hobbies, StudentEmails)
- This relation was in 3NF, we removed MVDs by decomposing into two tables to make it to 4NF.
- In 4NF, a table CANNOT have two or more independent multivalued attributes.
- The table R(StudentID, Hobbies, StudentEmails) had two independent multi-valued attributes i.e., Hobbies and StudentEmails, hence R was not in 4NF.
- We made separate tables for each multi-valued attribute to put it into 4NF.

How to convert a 3NF relation to 4NF?

- Think of what we did in the example of R(StudentID, Hobbies, StudentEmails)
- We just put the multi-valued attributes into separate tables

Learning point: To convert a relation from 3NF to 4NF, put the multi-valued attributes into separate tables

Good practice: Double-check to verify that no table contains two (or more) independent multi-valued attributes

Quick question

- Suppose there is a relation R (EmployeeID, Departments, CreditCardNo.)
- Assume that the same employee can belong to multiple departments and he can have more than 1 credit card.
- Suppose employeeID E001 belongs to three departments D1, D2, D3 and he has 2 credit cards numbered as C1 and C2.
- Which are the multi-valued attributes here?
  - Departments and CreditCardNo.
- Are these multi-valued attributes independent or are they related?
  - They are independent because whichever department he works for has nothing to do with his personal credit card numbers.

Quick question

- How would you convert this to 4NF?
  - First check if R is in 3NF
  - Is R in 3NF?
    - YES! (atomic values, no partial dependencies, no transitive dependencies)
  - Are two or more independent multi-valued attributes in the same relation?
    - YES! Hence, it violates 4NF, you need to convert it to 4NF

Answer

EmployeeID	DepartmentID
E001	D1
E001	D2
E001	D3

EmployeeID	CreditCardNo
E001	C1
E001	C2

We just separated the independent multi-valued attributes DepartmentID and CreditCardNo into different tables

Denormalization

- There is a trade-off between the extent of normalization and processing requirements
- When you do normalization, number of tables generally increases
  - Hence, chances are that you will have to query multiple tables to retrieve the data
    - This implies more disk I/Os and increases query execution time because of join operations across more tables
- Bottomline: Sometimes, you need to do denormalization for ensuring faster query execution times

Denormalization

- To address this trade-off effectively, think of the advantages of normalization
  - Normalization is good because it facilitates redundancies and helps in ensuring that anomalies do not arise
- Then think of what kind of queries are usual in the application scenario that you are dealing with
  - If due to normalization, the queries have to join across too many tables (thereby increasing query execution time), decide how much you want to denormalize based on gain in query execution time vs the loss in terms of anomalies arising later on
- When you perform any denormalization, be very careful that you note down what possible anomalies may arise later on due to the denormalization

Summary

- Normalization encourages good database design by facilitating the removal of anomalies and reducing data redundancies
- Remember that the normalization rules are only guidelines and it is ok to sometimes avoid using these guidelines for reducing query execution times
- For good database design, you need normalization + an appreciation of query processing requirements
  - Normalization alone is not sufficient for good database design