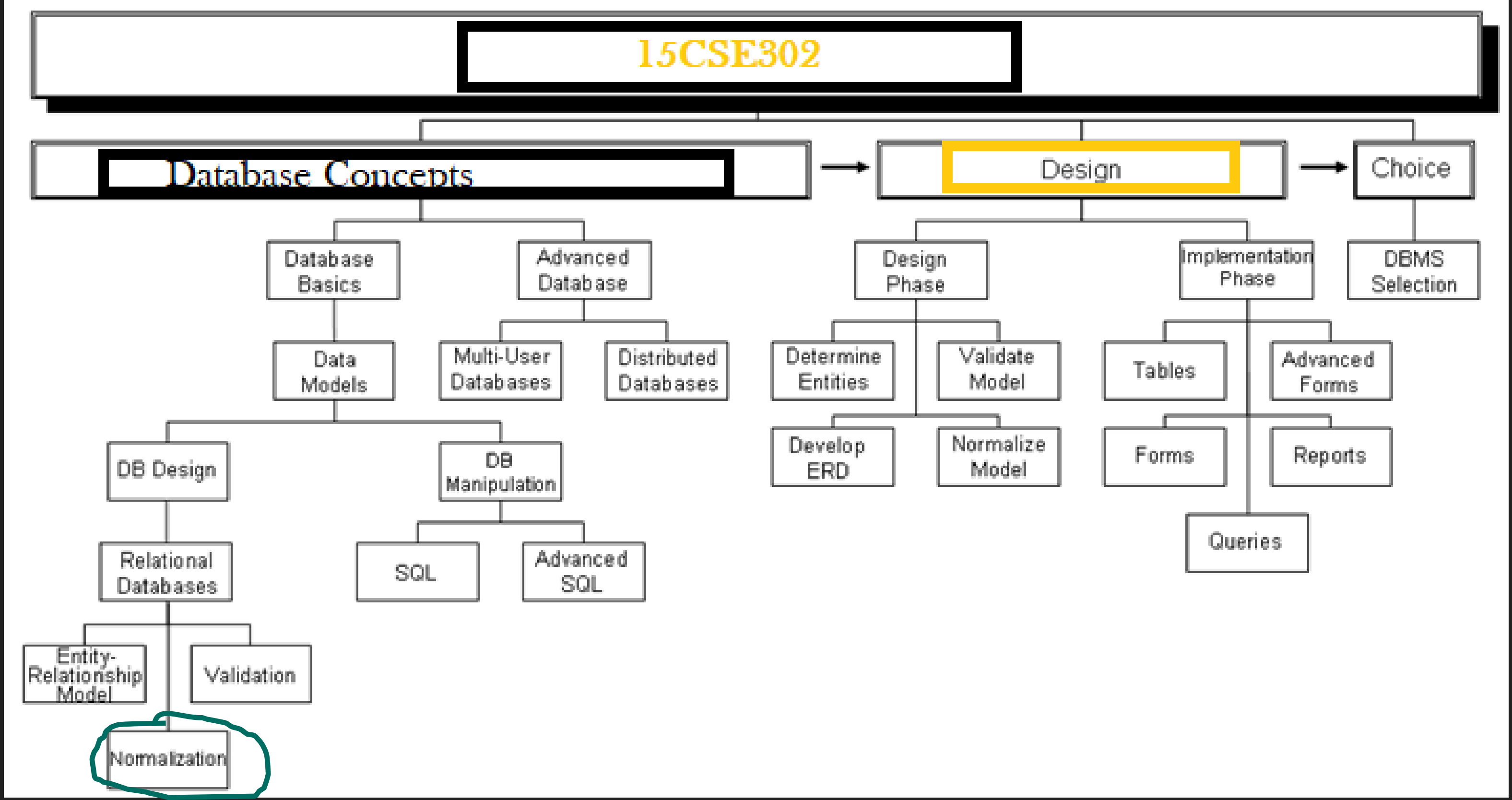
15CSE302 Database Management Systems
Lecture 25 Boyce-Codd Normal Form(BCNF)

B.Tech /III Year CSE/V Semester L T P C 2 0 2 3

DBMS Team
Dr G Jeyakumar
Bindu K R
Dr Priyanka Kumar
R. Manjusha
Department of CSE
Amrita School of Engineering

Syllabus



15CSE302
Database Concepts
Design
Choice
Database Basics
Advanced Database
Design Phase
Implementation Phase
DBMS Selection
Data Models
Multi-User Databases
Distributed Databases
Determine Entities
Validate Model
Tables
Advanced Forms
Develop ERD
Normalize Model
Forms
Reports
DB Design
DB Manipulation
Queries
Relational Databases
SQL
Advanced SQL
Entity-Relationship Model
Validation
Normalization

Brief Recap of Previous Lecture

- Dependency Preserving

Today we’ll discuss

BCNF

Desirable properties of Decomposition

When we decompose a relation schema R with a set of functional dependencies F into R_1, R_2, ..., R_n ,these properties should be satisfied

- **Non-additive (Lossless) join decomposition:**
  - Otherwise decomposition would result in information loss.
- **Dependency preservation:**
  - Otherwise, checking updates for violation of functional dependencies may require computing joins, which is expensive.
- **No redundancy:**

  The relations R_i preferably should be in either Boyce-Codd Normal Form or 3NF.

Boyce-Codd Normal Form

BCNF -Definition

- A relation schema R is in BCNF with respect to a set F if:
- For all functional dependencies of F of the form $\alpha \rightarrow \beta$, where $\alpha \subseteq R$ and $\beta \subseteq R$
  - $\alpha \rightarrow \beta$ is a trivial functional dependency($\beta \subseteq \alpha$)
  - α is a superkey for schema R
- A database design is in BCNF if each member of the set of relational schemas that constitute the design is in BCNF

Trivial Functional Dependency

- **Trivial** − If a functional dependency (FD) $X \rightarrow Y$ holds, where Y is a subset of X, then it is called a trivial FD. Trivial FDs always hold.
- **Non-trivial** − If an FD $X \rightarrow Y$ holds, where Y is not a subset of X, then it is called a non-trivial FD.
- **Completely non-trivial** − If an FD $X \rightarrow Y$ holds, where x intersect Y = Φ, it is said to be a completely non-trivial FD.

Trivial Functional Dependency

A functional dependency $X \rightarrow Y$ is trivial if Y is a subset of X

- {name, supervisor_id} $\rightarrow$ {name}
  - If two records have the same values on both the name and supervisor_id attributes, then they obviously have the same supervisor_id.
  - Trivial dependencies hold for all relation instances

A functional dependency $X \rightarrow Y$ is non-trivial if $Y \cap X = \emptyset$

- {supervisor_id} $\rightarrow$ {specialization}
  - Non-trivial FDs are given in the form of constraints when designing a database.
    - For instance, the specialization of a students must be the same as that of the supervisor.
  - They constrain the set of legal relation instances. For instance, if I try to insert two students under the same supervisor with different specializations, the insertion will be rejected by the DBMS

Some FDs are neither trivial nor non-trivial.

Boyce-Codd Normal Form

- **Eliminates all redundancy that can be discovered by functional dependencies**
- But, we can create a normal form more restrictive called 4NF

Rule for schema not in BCNF

- Let R be a schema not in BCNF, then there **is at least one nontrivial functional dependency** **$\alpha \rightarrow \beta$ such that α is not a superkey**

Example of not BCNF

- bor_loan = (customer_id, loan_number, amount)
- loan_number→amount

  but **loan_number** is not a superkey

BCNF Decomposition

- The definition of BCNF can be used to directly test if a relationship is in BCNF

- If a relation is not in BCNF it can be decomposed to create relations that are in BCNF

Example

- borrower = (customer_id, loan_number)

Is BCNF because no nontrivial functional dependency hold onto it

- loan = (loan_number, amount)

  Has one nontrivial functional dependency that holds,

  loan_number→amount

  but loan_number is a superkey so loan is in BCNF

3NF vs BCNF

- BCNF requires that all nontrivial dependencies be of the form $\alpha \rightarrow \beta$, where α is a superkey

- 3NF relaxes this constraint a little bit by allowing nontrivial functional dependencies

Testing for BCNF

- To check if a nontrivial dependency $\alpha \rightarrow \beta$ causes a violation of BCNF, compute a^+(attribute closure of α), and verify that it includes all attributes of *R*; that is, is is the superkey of *R*

- If we can show that none of the dependencies in *F* causes a violation of BCNF, then none of the dependencies in F^+ will cause a violation of BCNF either

Alternate test for 2

- For every subset α of attributes in R_i check that a^+(the attribute closure of α under F) either includes no attribute of $R_i - \alpha$, or includes all attributes of R_i

BCNF Decomposition Algorithm

- If R is not in BCNF, we can decompose R into a collection of BCNF schemas R_1, R_2, ..., R_n

```
Result := {R};
    done := false;
    computer F+
    while(not done) do
        if(there is a schema Ri in result that is not in BCNF)
                then begin
                        α→β be a nontrivial functional dependency that holds
                        on Ri such that α-> Ri is not in F+, and α∩ β=∅ ;
                        result :=(result – Ri) ∪ (Ri - β) ∪ (α,β);
                end
        else done := true;
```

Example

- lending = (branch_name, branch_city, assets, customer_name, loan_number, amount)
- branch_name→assets branch_city
- loan_number→amount branch_name
- candidate key is {loan_number, customer_name}
- branch_name is not superkey so not in BCNF so lending is not BCNF

Example (cont...)

- we replace lending by:

branch = (branch_name, branch_city, assets)
loan_info = (branch_name, customer_name, loan_number, amount)

- The only nontrivial functional dependencies that hold on branch include branch_name on the left side of the arrow.

- Since branch_name is a key for branch, the relation branch is in BCNF

Example (cont...)

- For loan_info

The functional dependency

loan_number→amount branch_name

holds on loan_info but loan_number is not a key for loan_info, so we replace loan_info by

loanb = (loan_number, branch_name, amount)

borrower = (customer_name, loan_number)

loanb and borrower are in BCNF

3NF advantages and disadvantages

- **Advantage of 3NF**: it is always possible to obtain a 3NF design without sacrificing losslessness or dependency preservation

- **Disadvantage of 3NF**: we may have to use null values to represent some of the possible meaningful relationships among data items, and there is the problem of repetition of information

BCNF vs 3NF

<u>BCNF</u>: For every functional dependency X->Y in a set F of functional dependencies over relation R, either:

- Y is a subset of X or,
- X is a superkey of R

<u>3NF</u>: For every functional dependency X->Y in a set *F* of functional dependencies over relation *R*, either:

- Y is a subset of X or,
- **X is a *superkey* of R, or**
- **Y is a subset of K for some key K of R**
- ***Note:* no subset of a key is a key**

3NF Schema

Client, Office -> Client, Office, Account
Account -> Office

Account	Client	Office
A	Joe	1
B	Mary	1
A	John	1
C	Joe	2

For every functional dependency X->Y in a set *F* of functional dependencies over relation *R*, either:

- Y is a subset of X or,
- X is a *superkey* of R, or
- Y is a subset of K for some key K of R

3NF Schema

Client, Office -> Client, Office, Account
Account -> Office

Account	Client	Office
A	Joe	1
B	Mary	1
A	John	1
C	Joe	2

For every functional dependency X->Y in a set *F* of functional dependencies over relation *R*, either:

- Y is a subset of X or,
- X is a *superkey* of R, or
- Y is a subset of K for some key K of R

BCNF Schema

Client, Office -> Client, Office, Account
Account -> Office

Account	Client	Office
A	Joe	1
B	Mary	1
A	John	1
C	Joe	2

For every functional dependency X->Y in a set *F* of functional dependencies over relation *R*, either:

Y is a subset of X or,
X is a superkey of R
Y is a subset of K

- 3NF has some redundancy BCNF does not
- Unfortunately, BCNF is not dependency preserving, but 3NF is.

Account	Office
A	1
B	1
C	2

Account	Client
A	Joe
B	Mary
A	John
C	Joe

Lossless decomposition

BCNF

For every functional dependency X->Y in a set *F*
of functional dependencies over relation *R*, either:
Y is a subset of X or, X is a superkey of R, Y is a subset of K

Closure(X):
c = X
Repeat
old = c
if there is an FD Z->V such that
$Z \subset c$ and
$V \not\subset c$ then
c = c U V
until old = c
return c

BCNFify(schema R, functional dependency set F):
D = {{R,F}}
while there is a schema S with dependencies F' in D that is not in BCNF, do:
given X->Y as a BCNF-violating FD in F
such that XY is in S
replace S in D with
S1={XY,F1} and
S2={(S-Y) U X, F2}
where F1 and F2 are the FDs in F over S1 or S2
(may need to split some FDs using decomposition)
End
return D

Reference

- Chen, Y. (2005). “Decomposition”. Retrieved on March 21, 2010 from http://www.cs.sjsu.edu/faculty/lee/cs157/Decomposition_YuHungChen.ppt
- Kamel, A. “Chapter 11Relational Database Design Algorithms” Retrieved on March 22, 2010 from http://www.cord.edu/faculty/kamel/08F-330/Presentations/ch11.pdf
- Lee, S. “Huffman code and Lossless Decomposition”. Retrieved on March 21, 2010 from http://www.cs.sjsu.edu/~lee/cs157b/29SpCS157BL14HuffmanCode&LosslessDecomposition.ppt
- Zaiane, O. (1998) “Dependency Preservation”. Retrieved on March 21, 2010 from http://www.cs.sfu.ca/CC/354/zaiane/material/notes/Chapter7/node8.html

Summary

- BCNF
- BCNF vs 3NF

Next Lecture

Normalization examples

Thank You

Happy to answer any questions ! ! !