>>

Normalisation

∧ >>

❖ Normalisation

Normalisation aims to put a schema into *xNF*

- by ensuring that all relations in the schema are in *xNF*

How normalisation works ...

- decide which normal form *xNF* is "acceptable"
  - i.e. how much redundancy are we willing to tolerate?
- check whether each relation in schema is in *xNF*
- if a relation is not in *xNF*
  - partition into sub-relations where each is "closer to" *xNF*
- repeat until all relations in schema are in *xNF*

<< ∧ >>

❖ Normalisation (cont)

In practice, BCNF and 3NF are the most important normal forms.

Boyce-Codd Normal Form (BCNF):

- eliminates all redundancy due to functional dependencies
- but may not preserve original functional dependencies

Third Normal Form (3NF):

- eliminates most (but not all) redundancy due to *fd*s
- guaranteed to preserve all functional dependencies

<< ∧ >>

❖ Relation Decomposition

The standard transformation technique to remove redundancy:

- decompose relation R into relations S and T

We accomplish decomposition by

- selecting (overlapping) subsets of attributes
- forming new relations based on attribute subsets

Properties: $R = S \cup T$, $S \cap T \neq \emptyset$ and $r(R) = s(S) \bowtie t(T)$

May require several decompositions to achieve acceptable NF.

Normalisation algorithms tell us how to choose S and T.

❖ Schema (Re)Design

Consider the following relation for *BankLoans*:

branchName	branchCity	assets	custName	loanNo	amount
Downtown	Brooklyn	9000000	Jones	L-17	1000
Redwood	Palo Alto	2100000	Smith	L-23	2000
Perryridge	Horseneck	1700000	Hayes	L-15	1500
Downtown	Brooklyn	9000000	Jackson	L-15	1500
Mianus	Horseneck	400000	Jones	L-93	500
Round Hill	Horseneck	8000000	Turner	L-11	900
North Town	Rye	3700000	Hayes	L-16	1300

This schema has all of the update anomalies mentioned earlier.

<< ^ >>

❖ Schema (Re)Design (cont)

To improve the design, decompose the *BankLoans* relation.

The following decomposition is not helpful:

Branch(branchName, branchCity, assets)
CustLoan(custName, loanNo, amount)

because we lose information (which branch is a loan held at?)

Another possible decomposition:

BranchCust(branchName, branchCity, assets, custName)
CustLoan(custName, loanNo, amount)

<< ^ >>

❖ Schema (Re)Design (cont)

The *BranchCust* relation instance:

branchName	branchCity	assets	custName
Downtown	Brooklyn	9000000	Jones
Redwood	Palo Alto	2100000	Smith
Perryridge	Horseneck	1700000	Hayes
Downtown	Brooklyn	9000000	Jackson
Mianus	Horseneck	400000	Jones
Round Hill	Horseneck	8000000	Turner
North Town	Rye	3700000	Hayes

❖ Schema (Re)Design (cont)

The *CustLoan* relation instance:

custName	loanNo	amount
Jones	L-17	1000
Smith	L-23	2000
Hayes	L-15	1500
Jackson	L-15	1500
Jones	L-93	500
Turner	L-11	900
Hayes	L-16	1300

❖ Schema (Re)Design (cont)

Now consider the result of *(BranchCust* ⋈ *CustLoan)*

branchName	branchCity	assets	custName	loanNo	amount
Downtown	Brooklyn	9000000	Jones	L-17	1000
Downtown	Brooklyn	9000000	Jones	L-93	500
Redwood	Palo Alto	2100000	Smith	L-23	2000
Perryridge	Horseneck	1700000	Hayes	L-15	1500
Perryridge	Horseneck	1700000	Hayes	L-16	1300
Downtown	Brooklyn	9000000	Jackson	L-15	1500
Mianus	Horseneck	400000	Jones	L-93	500
Mianus	Horseneck	400000	Jones	L-17	1000
Round Hill	Horseneck	8000000	Turner	L-11	900
North Town	Rye	3700000	Hayes	L-16	1300
North Town	Rye	3700000	Hayes	L-15	1500

<< ∧ >>

❖ Schema (Re)Design (cont)

This is clearly not a successful decomposition.

The fact that we ended up with extra tuples was symptomatic of losing some critical "connection" information during the decomposition.

Such a decomposition is called a lossy decomposition.

In a good decomposition, we should be able to reconstruct the original relation exactly:

if R is decomposed into S and T, then $S \bowtie T = R$

Such a decomposition is called lossless join decomposition.

❖ BCNF Normalisation Algorithm

The following algorithm converts an arbitrary schema to BCNF:

```
Inputs: schema R, set F of fds
Output: set Res of BCNF schemas

Res = {R};
while (any schema S ∈ Res is not in BCNF) {
    choose any fd X → Y on S that violates BCNF
    Res = (Res-S) ∪ (S-Y) ∪ XY
}
```

The last step means: make a table from *XY*; drop *Y* from table *S*

The "choose any" step means that the algorithm is non-deterministic

❖ BCNF Normalisation Example

Recall the *BankLoans* schema:

BankLoans(branchName, branchCity, assets, custName, loanNo, amount)

Rename to simplify ...

B = branchName, C = branchCity, A = assets, N = CustName, L = loanNo, M = amount

So ... $R = BCANLM$, $F = \{ B \rightarrow CA, L \rightarrow MN \}$, $key(R) = BL$

R is not in BCNF, because $B \rightarrow CA$ is not a whole key

Decompose into

- $S = BCA$, $F_S = \{ B \rightarrow CA \}$ $key(S) = B$
- $T = BNLM$, $F_T = \{ L \rightarrow NM \}$, $key(T) = BL$

(continued)

<< ∧ >>

❖ BCNF Normalisation Example (cont)

$S = BCA$ is in BCNF, only one fd and it has key on LHS

$T = BLNM$ is not in BCNF, because $L \rightarrow NM$ is not a whole key

Decompose into ...

- $U = LNM$, $F_U = \{ L \rightarrow NM \}$, $key(U) = L$, which is BCNF
- $V = BL$, $F_V = \{\}$, $key(V) = BL$, which is BCNF

Result:

- $S = (branchName, branchCity, assets) = Branches$
- $U = (loanNo, custName, amount) = Loans$
- $V = (branchName, loanNo) = BranchOfLoan$

❖ 3NF Normalisation Algorithm

The following algorithm converts an arbitrary schema to 3NF:

```
Inputs: schema R, set F of fds
Output: set Ri of 3NF schemas

let Fc be a reduced minimal cover for F
Res = {}
for each fd X → Y in Fc {
    if (no schema S ∈ Res contains XY) {
        Res = Res ∪ XY
    }
}
if (no schema S ∈ Res contains a key for R) {
    K = any candidate key for R
    Res = Res ∪ K
}
```

<< ∧ >>

❖ 3NF Normalisation Example

Recall the *BankLoans* schema:

BankLoans(branchName, branchCity, assets, custName, loanNo, amount)

Rename to simplify ...

$R = BCANLM$, $F = \{ B \rightarrow CA, L \rightarrow MN \}$, $key(R) = BL$

Compute minimal cover = $\{ B \rightarrow C, B \rightarrow A, L \rightarrow M, L \rightarrow N \}$

Reduce minimal cover = $\{ B \rightarrow CA, L \rightarrow MN \}$

Convert into relations: $S = BCA$, $T = LNM$

No relation has key BL, so add new table containing key $U = BL$

Result is $S = BCA$, $T = LNM$, $U = BL$ ... same as BCNF

<< ^

❖ Database Design Methodology

To achieve a "good" database design:

- identify attributes, entities, relationships → ER design
- map ER design to relational schema
- identify constraints (including keys and functional dependencies)
- apply BCNF/3NF algorithms to produce normalised schema

Note: may subsequently need to "denormalise" if the design yields inadequate performance.

Produced: 5 Nov 2020