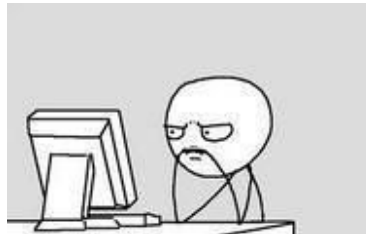


Announcements

1. XQuery extra **tutorial** this week!
 - ▶ 1st option: Thursday 5-7pm (MC 252)
 - ▶ 2nd option: Friday 10am-12pm (GB 220)
2. Course **evaluations** are available online
 - ▶ Your feedback is very important!
 - ▶ Tell us what you liked, what you didn't like



3. A3 is out, due Dec 9
 - ▶ Xquery (Part1); FDs and Decomposition (Part2)

Design Theory for Relational Databases Part II

csc343, Fall 2015

Based on slides from Diane Horton;
Originally based on slides by Jeff Ullman.

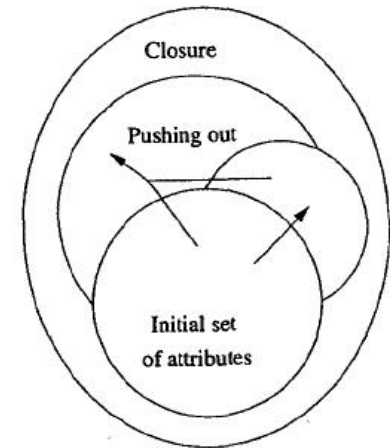
Recap

✓ Functional Dependencies (FD)

➤ $A \rightarrow B$ (any 2 tuples that agree on A must agree on B)

✓ Closure

➤ “ A^+ ” = everything I can determine from A, using a given set of FDs



✓ FD Projection

➤ Map FDs from one relation to another (smaller) relation

✓ Minimal Cover

➤ An equivalent set of FDs that contain no redundancies and can't be further reduced

◆ **Today:** Normalization: BCNF, 3NF

Minimal Basis – Last week's example

- ◆ What is the minimal cover of these FDs in ABCDEG:

$A \rightarrow B$, $ABCD \rightarrow E$, $G \rightarrow A$, $G \rightarrow B$

Answer:

(1) Can any of the FDs be implied from another FD? (if so, **drop**)

***Systematic approach:**

- Calculate the closure of the LHS in each FD, using the rest of FDs;
Can we reach the RHS using the other FDs?

1. $A \rightarrow B$

A^+ under $\{2,3,4\}$? = ...

2. $ABCD \rightarrow E$

$ABCD^+$ under $\{1,3,4\}$? = ...

3. $G \rightarrow A$

G^+ under $\{1,2,4\}$? = ...

4. $G \rightarrow B$

G^+ under $\{1,2,3\}$? = **GAB**

- **Drop FD#4.**

Minimal Basis – Last week's example

- ◆ What is the minimal cover of these FDs in ABCDEG:

$$A \rightarrow B, \quad ABCD \rightarrow E, \quad G \rightarrow A, \quad \cancel{G \rightarrow B}$$

Answer:

(2) Check if any LHS can be reduced (any attributes can be removed?). If so, **drop** the extra attributes.

1. $A \rightarrow B$

2. ~~$ABCD \rightarrow E$~~

Start with removing 1 attribute.. (ACD+, ABC+, BCD+, ..and so on)

$$ACD^+ = ABCD$$

2. $ACD \rightarrow E$

3. $G \rightarrow A$

➤ Result: Minimal basis is $A \rightarrow B, \quad ACD \rightarrow E, \quad G \rightarrow A$

Minimal Basis Steps – Comment

◆ Step #1 is actually:

Make all RHS singleton (one attribute)

Example:

1. $CB \rightarrow ED$

becomes..

1. $CB \rightarrow E$

2. $CB \rightarrow D$

Some comments on computing a minimal basis

- ◆ Often there are multiple possible results, depending on the order in which you consider the possible simplifications.
- ◆ After you identify a **redundant** FD, you must **not** use it when computing any subsequent closures (as you consider whether other FDs are redundant).

... and some that are less intuitive

- ◆ When you are computing closures to decide whether the LHS of an FD

$$a_1 a_2 \dots a_m \rightarrow b_1 b_2 \dots b_n$$

can be simplified, continue to use that FD.

- ◆ When you have tried to eliminate each FD and to reduce each LHS, you must go back and try again.

Part II:

Using FD Theory to do Database Design

Recall that poorly designed table?

part	manufacturer	manAddress	seller	sellerAddress	price
1983	Hammers `R Us	99 Pinecrest	ABC	1229 Bloor W	5.59
8624	Lee Valley	102 Vaughn	ABC	1229 Bloor W	23.99
9141	Hammers `R Us	99 Pinecrest	ABC	1229 Bloor W	12.50
1983	Hammers `R Us	99 Pinecrest	Walmart	5289 St Clair W	4.99

- ◆ We can now express the relationships as FDs:
 - ◆ $\text{part} \rightarrow \text{manufacturer}$
 - ◆ $\text{manufacturer} \rightarrow \text{manAddress}$
 - ◆ $\text{seller} \rightarrow \text{sellerAddress}$
- ◆ These FDs point to a bad design
- ◆ To know how, let's first talk about decomposition.

Decomposition

- ◆ To improve a badly-designed schema $R(A_1, A_2, \dots, A_n)$, we will decompose it into smaller relations

$S(B_1, B_2, \dots, B_m)$ and $T(C_1, C_2, \dots, C_k)$ such that:

- ◆ $S = \pi_{B_1, B_2, \dots, B_m}(R)$

- ◆ $T = \pi_{C_1, C_2, \dots, C_k}(R)$

- ◆ $\{A_1, A_2, \dots, A_n\} = \{B_1, B_2, \dots, B_m\} \cup \{C_1, C_2, \dots, C_k\}$

But *which* decomposition?

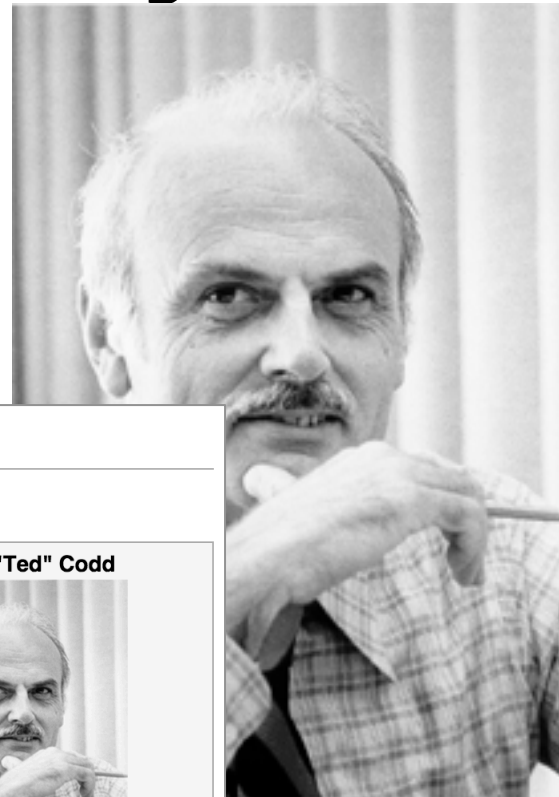
- ◆ Decomposition can definitely improve a schema.
- ◆ But which decomposition?
There are many possibilities.
- ◆ And how can we be sure a new schema doesn't exhibit other anomalies?
- ◆ **Boyce-Codd Normal Form** *guarantees* it.

Boyce-Codd

Raymond Boyce



Edgar Codd



Edgar F. Codd

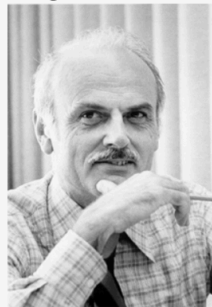
From Wikipedia, the free encyclopedia

Edgar Frank "Ted" Codd (19 August 1923 – 18 April 2003) was an English [computer scientist](#) who, while working for [IBM](#), invented the [relational model](#) for [database](#) management, the theoretical basis for [relational databases](#). He made other valuable contributions to [computer science](#), but the relational model, a very influential general theory of data management, remains his most mentioned achievement.^{[6][7]}

Contents [\[hide\]](#)

- [1 Biography](#)
- [2 Work](#)
- [3 Publications](#)
- [4 See also](#)

Edgar "Ted" Codd



Born

Edgar Frank Codd
19 August 1923^{[1][2]}

Boyce-Codd Normal Form

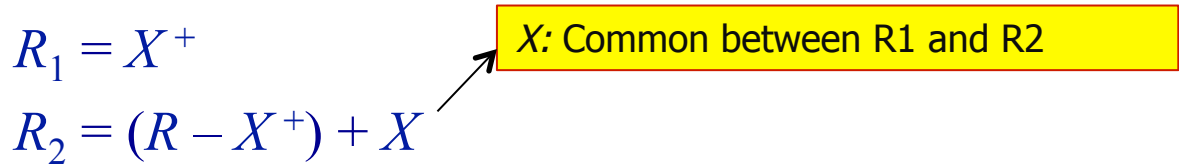
- ◆ We say a relation R is in **BCNF** if: for every nontrivial FD $X \rightarrow Y$ that holds in R , X is a **superkey**.
 - ◆ Remember: *nontrivial* means Y is not contained in X .
 - ◆ Remember: a *superkey* doesn't have to be minimal.
- ◆ In other words, BCNF requires that:
“Only things that FD *everything*, can FD anything.”
- ◆ **Exercise:** Suppose we have a relation **Students**(SID, email, course, term, prof) and that these FDs hold:
 $\{ \text{SID} \rightarrow \text{email}; \text{course, term} \rightarrow \text{prof}; \text{SID, course} \rightarrow \text{grade} \}$
Is this relation in **BCNF**?
Solution: Test if the LHS in each FD is a superkey
(1) $\text{SID}^+ = \text{SID, email}$
(LHS not a superkey, then R is not in BCNF – no need to check other FDs)

R is a relation; F is a set of FDs.

Return the BCNF decomposition of R , given these FDs.

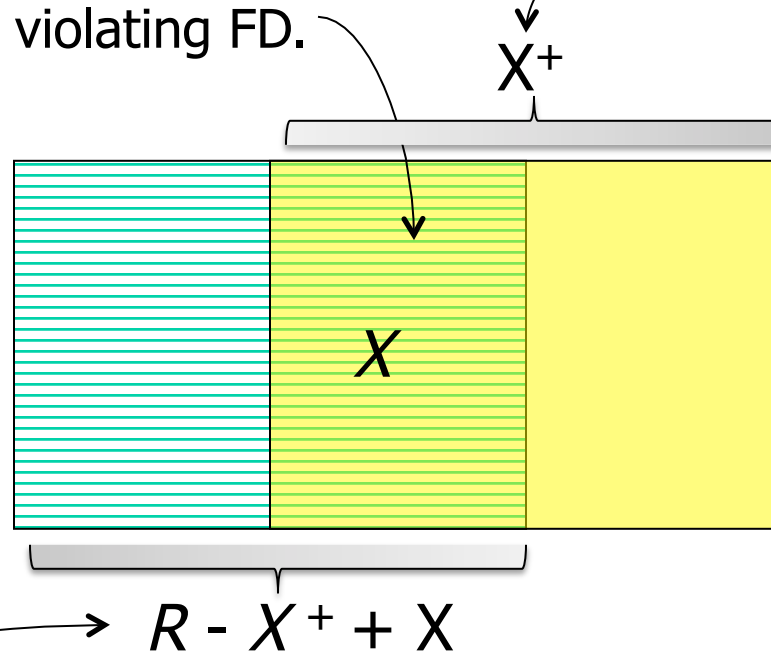
BCNF_decomp(R, F):

If an FD $X \rightarrow Y$ in F violates BCNF

1. Compute X^+ .
2. Replace R by two relations with schemas:
 $R_1 = X^+$
 $R_2 = (R - X^+) \cup X$

3. Project the FD's F onto R_1 and R_2 .
4. Recursively decompose R_1 and R_2 into BCNF.

Decomposition Picture

- 1) Start with the LHS of the violating FD.
- 2) Close the LHS to get one new relation



- 3) Everything except the new stuff is the other new relation.
 X is in both new relations to make a connection between them.

Some comments on BCNF decomp

- ◆ If more than one FD violates BCNF, you may decompose based on any one of them.
 - ▶ Because of this, there may be multiple possible results.
- ◆ The new relations we create may not be in BCNF. We must recurse.
 - ▶ We only keep the relations at the “leaves”.
- ◆ How does the decomposition step help?
 - ▶ Exercise..

Example – BCNF Decomposition

- ◆ Suppose we have the following FDs for $R(A,B,C,D,E)$:

$$A \rightarrow B, \quad CD \rightarrow E$$

Apply BCNF Decomposition.

Iteration #1: R

Consider $A \rightarrow B$ (Is the LHS a superkey?)

➤ $A^+ = AB$

A is not a superkey! (Violates BCNF)

➤ Decompose:

- ~~$R_1=AB, R_2=CDE$~~
- $R_1=AB, R_2=ACDE$

➤ Project FDs:

$R_1(A,B): A \rightarrow B$ (A is a superkey!) ✓ R1 in BCNF

$R_2(A,C,D,E): CD \rightarrow E$ (CD is **not** a superkey)

Example – BCNF Decomposition

- ◆ Suppose we have the following FDs for $R(A,B,C,D,E)$:

$$A \rightarrow B, \quad CD \rightarrow E$$

Answer (contd): $R1(AB)$ in BCNF .. $R2(ACDE)$ is not

Iteration #2: Decompose $R2$

$R2(A,C,D,E)$: $CD \rightarrow E$

- CD is not a superkey ($CD \rightarrow E$ violates BCNF)

$CD^+ = CDE$

- Decompose to:

- $R3=CDE$, $R4=ACD$

- Project FDs:

$R3(C,D,E)$: $CD \rightarrow E$ (CD is a superkey!)

✓ **$R3$ doesn't violate BCNF**

$R4(A,C,D)$: No FDs. Key: ACD

✓ **$R4$ doesn't violate BCNF**

•BCNF Decomposition Result: **$R1(A,B)$, $R3(C,D,E)$, $R4(A,C,D)$**

Speed-ups for BCNF decomposition

- ◆ Don't need to know *all* superkeys.
 - ▶ Only need to check whether the **LHS** of each **FD** is a superkey.
 - ▶ Use the **closure test** (simple and fast!)
- ◆ When projecting FDs onto a new relation, check each new FD:
 - ▶ Does the new relation violate BCNF because of this FD?
 - ▶ If so, **abort** the projection.

You are about to discard this relation anyway (and decompose further).

Properties of Decompositions

What we want from a decomposition

1. No anomalies.

2. Lossless Join : It should be possible to

a) project the original relations onto the decomposed schema

b) then reconstruct the original by joining.

We should get back **exactly the original tuples.**

3. Dependency Preservation :

All the original FD's should be satisfied.

What is lost in a “lossy” join?

◆ For any decomposition, it is the case that:

► $r \subseteq r_1 \bowtie \dots \bowtie r_n$

► I.e., we will get back every tuple.

◆ But it may *not* be the case that:

► $r \supseteq r_1 \bowtie \dots \bowtie r$

► I.e., we can get spurious tuples.

R

cdf	name	grade
g3tout	Amy	91
g4foobar	David	78
c0zhang	David	85

R1

cdf	name
g3tout	Amy
g4foobar	David
c0zhang	David

R2

name	grade
Amy	91
David	78
David	85

R1 ⋈ **R2**

cdf	name	grade
g3tout	Amy	91
g4foobar	David	78
g4foobar	David	85
c0zhang	David	78
c0zhang	David	85

X

What **BCNF decomposition** offers

1. *No anomalies* : ✓ (Due to no redundancy)
2. *Lossless Join* : ✓ (Section 3.4.1 argues this)
3. *Dependency Preservation* : ✗

Warning The BCNF *property* does *not* guarantee lossless join

- ◆ If you use the BCNF *decomposition* algorithm, then yes a lossless join is guaranteed!
- ◆ If you generate a decomposition some other way
 - ◆ you have to check to make sure you have a lossless join
 - ◆ even if your schema satisfies BCNF!
- ◆ We'll learn an algorithm for this check later.

Preservation of dependencies

- ◆ BCNF decomposition does not guarantee preservation of dependencies.
- ◆ I.e., in the schema that results, it may be possible to create an instance that:
 - ◆ satisfies all the FDs in the final schema,
 - ◆ but violates one of the original FDs.
- ◆ Why? Because the algorithm goes too far — breaks relations down too much.

A less strict version of BCNF: 3NF

- ◆ *3rd Normal Form* (3NF) modifies the BCNF condition to be less strict.
- ◆ Def: an attribute is *prime* if it is a member of any key.
- ◆ $X \rightarrow A$ is in 3NF if X is a superkey, OR, if A is prime.
- ◆ I.e., it's ok if X is not a superkey, as long as A is prime!
- ◆ [Exercise]

F is a set of FDs; L is a set of attributes.
Synthesize and return a schema in 3rd Normal Form.

3NF_synthesis(F, L):

Construct a minimal basis M for F.

For each FD $X \rightarrow Y$ in M

Define a new relation with schema $X \cup Y$.

If no relation is a superkey for L

Add a relation whose schema is some key.

Simple Example – 3NF Synthesis

- ◆ Suppose we have the following FDs for $R(A,B,C,D,E)$:

$$A \rightarrow B, \quad CD \rightarrow E$$

3NF Synthesis:

- Compute all keys for R

Keys: { ACD }

- Find the Minimal Cover

MC: { $A \rightarrow B, \quad CD \rightarrow E$ }

- Use FDs in Minimal Basis to define new relations

$R_1(A,B), \quad R_2(C,D,E)$

- If none are a superkey for R, add relation whose schema is some key

$R_1(A,B), \quad R_2(C,D,E)$

Add $R_3(A,C,D)$

****PLEASE SEE MORE EXAMPLES ON COURSE WEBSITE****

3NF synthesis doesn't "go too far"

- ◆ BCNF decomposition doesn't stop decomposing until in all relations:
 - ▶ if $X \rightarrow A$ then X is a superkey.
- ◆ 3NF generates relations where:
 - ▶ $X \rightarrow A$ and yet X is *not* a superkey, but A is at least prime.

What a 3NF decomposition offers

1. No anomalies : ✗

2. Lossless Join : ✓

3. Dependency Preservation : ✓

- ◆ Neither BCNF nor 3NF can guarantee all three!
We must be satisfied with 2 of 3.
- ◆ Decompose too far \Rightarrow can't enforce all FDs.
- ◆ Not far enough \Rightarrow can have redundancy.
- ◆ We consider a schema “good” if it is in either **BCNF** or **3NF**.

How can we get anomalies in 3NF?

- ◆ 3NF synthesis guarantees that the resulting schema will be in 3rd normal form.
- ◆ This allows FDs with a **non-superkey** on the LHS.
- ◆ This allows redundancy, and thus anomalies.

“Synthesis” vs “decomposition”

◆ 3NF synthesis:

- ▶ We build up the relations in the schema from nothing.

◆ BCNF decomposition:

- ▶ We start with a bad relation schema and break it down.

Testing for a Lossless Join

- ◆ If we project R onto R_1, R_2, \dots, R_k , can we recover R by rejoining?
- ◆ We will get all of R .
 - ▶ Any tuple in R can be recovered from its projected fragments. This is guaranteed.
- ◆ But will we get **only** R ?
 - ▶ Can we get a tuple we didn't have in R ?
This part we must check.

Aside: when we don't need to test for lossless Join

- ◆ Both BCNF decomposition and 3NF synthesis guarantee lossless join.
- ◆ So we never need to test for lossless join if we have done BCNF decomposition or 3NF synthesis.
- ◆ But merely satisfying BCNF or 3NF does not guarantee a lossless join!

The Chase Test

- ◆ An organized way to see if a tuple t in the natural join of subschemas R_i to be a tuple in original R , using the FDs.
- ◆ Suppose tuple t appears in the join.
- ◆ Then t is the join of projections of some tuples of R , one for each R_i of the decomposition.
- ◆ Can we use the given FD's to show that one of these tuples must be t ?



Setup for the Chase Test

- ◆ Start by assuming $t = abc... .$
- ◆ For each i , there is a tuple s_i of R that has $a, b, c,...$ in the attributes of R_i .
- ◆ s_i can have any values in other attributes.
- ◆ We'll use the same letter as in t , but with a subscript, for these components.

The algorithm

1. If two rows agree in the left side of a FD, make their right sides agree too.
2. Always replace a subscripted symbol by the corresponding unsubscripted one, if possible.
3. If we ever get a completely unsubscripted row, we know any tuple in the project-join is in the original (*i.e.*, the join is lossless).
4. Otherwise, the final tableau is a counterexample (*i.e.*, the join is lossy).

Example: The Chase

- ◆ Let $R = ABCD$, and the decomposition be AB , BC , and CD .
- ◆ Let the given FD's be $C \rightarrow D$ and $B \rightarrow A$.
- ◆ Suppose the tuple $t = abcd$ is the join of tuples projected onto AB , BC , CD .

- Suppose the tuple $t = abcd$ is the join of tuples projected onto *AB*, *BC*, *CD*.

FDs:

$C \rightarrow D, B \rightarrow A$

The tuples
of R pro-
jected onto
 AB, BC, CD .

The *Tableau*

	A	B	C	D
R1(AB)	<i>a</i>	<i>b</i>	c_1	d_1
R2(BC)	a_2 <i>a</i>	<i>b</i>	<i>c</i>	d_2 <i>d</i>
R3(CD)	a_3	b_3	<i>c</i>	<i>d</i>

Use $B \rightarrow A$

We've proved the
second tuple must be t .

Use $C \rightarrow D$