A BCNF Normalisation Algorithm

Input:

- A *specification* containing:
  1. a *universal* set of attributes U, and
  2. a set of functional dependencies (FDs) F over U.
- An entity-relationship diagram (ERD) conforming to the specification.

Output: A database schema **R** which is in BCNF with respect to F.

A BCNF Normalisation Algorithm

Input:

- A *specification* containing:
  1. a *universal* set of attributes U, and
  2. a set of functional dependencies (FDs) F over U.
- An entity-relationship diagram (ERD) conforming to the specification.

Output: A database schema **R** which is in BCNF with respect to F.

Notes:

- **R** = $\{R_1, R_2, \ldots, R_n\}$, where each R_i is a subset of U.
- The union of all schema(R_i) is U.
- **R** is called a **decomposition** of U.

Strategy

Step 1. Convert ERD into a database schema **S**.

Step 2. If any of the relation schemas in **S** are *not* in BCNF with respect to F, then decompose them further, using the DECOMPOSE algorithm, given in this lecture.

Algorithm ERD-TO-BCNF(ERD, F)

1. *Convert ERD into a database schema* **S** = $\{S_1, \ldots, S_m\}$;
2. **let** *the output database schema* **R** *be empty;*
3. **for each** S_i *in* **S do**
4. **if** *TEST-BCNF*(S_i, *F) = YES*
5. **add** S_i *to* **R**;
6. **else**
7. **merge** *DECOMPOSE*(S_i, *F) and* **R**;
8. **end for**
9. **return** *the decomposition* **R**;

Testing whether a relation schema is in BCNF

Algorithm TEST-BCNF(R, F)

Assume F is a set of canonical FDs

1. **for each** *(non-trivial)* $X \rightarrow A$ *in* F^+ **do**
2. **if** X *is* **not** *a superkey with respect to* F
3. **return** *NO;*
4. **end if**
5. **end for**
6. **return** *YES;*

Testing whether a relation schema is in BCNF

Algorithm TEST-BCNF(R, F)

Assume F is a set of canonical FDs

1. **for each** *(non-trivial)* $X \rightarrow A$ *in* F^+ **do**
2. **if** X *is* **not** *a superkey with respect to* F
3. **return** *NO;*
4. **end if**
5. **end for**
6. **return** *YES;*

Note that, in general, we need to consider F^+, the closure of F, to check whether there are any FDs which violate BCNF.

But we can start trying to find violations in F, and only consider F^+ once we find no violations in F.

Example

- Let schema(PHONE) = {cust-name, phone-num, phone-network}.
- Let F = {cust-name $\rightarrow$ phone-num, phone-num $\rightarrow$ phone-network}.

Is PHONE in BCNF with respect to F ?

Decomposition Condition used by Algorithm

So $\boxed{\text{phone-num} \rightarrow \text{phone-network}}$ **violates** BCNF.

- phone-num is the **left-hand** side of the violating FD
- phone-network is the **right-hand** side of the violating FD

Split PHONE into two relation schemas:

Decomposition Condition used by Algorithm

So $\boxed{\text{phone-num} \rightarrow \text{phone-network}}$ **violates** BCNF.

- phone-num is the **left-hand** side of the violating FD
- phone-network is the **right-hand** side of the violating FD

Split PHONE into two relation schemas:

1. R_1 = NETWORK, with schema(NETWORK) = {phone-num, phone-network}, containing all the attributes in the **violating** FD, and

   F_1 = { phone-num $\rightarrow$ phone-network }.

Decomposition Condition used by Algorithm

So $\boxed{\text{phone-num} \rightarrow \text{phone-network}}$ **violates** BCNF.

- phone-num is the **left-hand** side of the violating FD
- phone-network is the **right-hand** side of the violating FD

Split PHONE into two relation schemas:

1. R_1 = NETWORK, with schema(NETWORK) = {phone-num, phone-network}, containing all the attributes in the **violating** FD, and

   F_1 = { phone-num $\rightarrow$ phone-network }.
2. R_2 = CUST, with schema(CUST) = {cust-name, phone-num}, containing the attributes in schema(PHONE) **except** those in the right-hand side of the **violating** FD, and

   F_2 = { cust-name $\rightarrow$ phone-num }.

Algorithm DECOMPOSE(R, F)
Assume F is a set of canonical FDs

1. **let** *the output database schema* **Out** *be empty;*
2. **if** *TEST-BCNF(R, F) = YES* **then**
3. **add** *R to* **Out**;
4. **else**
5. **let** *$X \rightarrow A$ in F^+ be nontrivial (i.e. A is* **not** *in X) such that X is* **not** *a superkey with respect to F;*
6. **let** *R_1 be a relation schema, with schema(R_1) = X* **merged** *with A;*
7. **merge** *DECOMPOSE(R_1, F) and* **Out**;
8. **let** *R_2 be a relation schema, with schema(R_2) = schema(R)* **except** *A;*
9. **merge** *DECOMPOSE(R_2, F) and* **Out**;
10. **end if**
11. **return Out**;

Result. DECOMPOSE(R, F) returns a decomposition of schema(R).

Result. The natural join can be applied to all of the relations in DECOMPOSE(R, F) to recover precisely the information stored in any relation over schema(R); this is known as the **lossless join** property.

Lossless join

Recall example: S is a relation schema, with schema(S) = {ENAME, CNAME, SAL} and single FD: ENAME $\rightarrow$ SAL

(Modified) relation s over S is given by

ENAME	CNAME	SAL
Jack	Diane	25
Jack	John	25
Donald	Diane	30
Donald	David	30

If we decompose S into {ENAME,CNAME} and {CNAME,SAL} as follows:

ENAME	CNAME
Jack	Diane
Jack	John
Donald	Diane
Donald	David

CNAME	SAL
Diane	25
John	25
Diane	30
David	30

If we decompose S into {ENAME,CNAME} and {CNAME,SAL} as follows:

ENAME	CNAME
Jack	Diane
Jack	John
Donald	Diane
Donald	David

CNAME	SAL
Diane	25
John	25
Diane	30
David	30

and then perform the natural join, we get

ENAME	CNAME	SAL
Jack	Diane	25
Jack	**Diane**	**30**
Jack	John	25
Donald	**Diane**	**25**
Donald	Diane	30
Donald	David	30

$\Rightarrow$ with two tuples that were *not* in the original relation

- A decomposition such as that into {ENAME,CNAME} and {CNAME,SAL} is called **lossy**
- We started knowing Jack's salary was 25
- After decomposing, if we query Jack's salary we get both 25 and 30
- The decomposition does not faithfully represent the original information we had

- A decomposition such as that into {ENAME,CNAME} and {CNAME,SAL} is called **lossy**
- We started knowing Jack's salary was 25
- After decomposing, if we query Jack's salary we get both 25 and 30
- The decomposition does not faithfully represent the original information we had
- A decomposition which *does* faithfully represent the original information is called **lossless**
- The lossless condition is guaranteed if we ensure that the common attributes between a pair of decomposed relation schemas is a key for one of them
- The BCNF algorithm ensures lossless decompositions

Example of BCNF Decomposition

Recall example involving employee names (ENAME), salaries (SAL) and children (CNAME). Let's call the relation schema EMP.

The assumed set of FDs was F_2 = { ENAME $\rightarrow$ SAL }.

- ENAME $\rightarrow$ SAL **violates** BCNF in EMP, so *decompose* EMP into

  ES, with schema(ES) = { ENAME, SAL }, and

  EC, with schema(EC) = { ENAME, CNAME }
- Both ES and EC are in BCNF:
  - ENAME is a key in ES
  - the only key for EC is (ENAME, CNAME)

Another Example of BCNF Decomposition

Let STUD be a relation schema, with schema(STUD) = {SNUM, POSTCODE, CITY, COUNTRY}, with FDs

{SNUM $\rightarrow$ POSTCODE, POSTCODE $\rightarrow$ CITY, CITY $\rightarrow$ COUNTRY}

Another Example of BCNF Decomposition

Let STUD be a relation schema, with schema(STUD) = {SNUM, POSTCODE, CITY, COUNTRY}, with FDs

{SNUM $\rightarrow$ POSTCODE, POSTCODE $\rightarrow$ CITY, CITY $\rightarrow$ COUNTRY}

- CITY $\rightarrow$ COUNTRY **violates** BCNF in STUD, so *decompose* STUD into

  CC, with schema(CC) = {CITY, COUNTRY}, and

  STUD1, with schema(STUD1) = {SNUM, POSTCODE, CITY}

Another Example of BCNF Decomposition

Let STUD be a relation schema, with schema(STUD) = {SNUM, POSTCODE, CITY, COUNTRY}, with FDs

{SNUM $\rightarrow$ POSTCODE, POSTCODE $\rightarrow$ CITY, CITY $\rightarrow$ COUNTRY}

- CITY $\rightarrow$ COUNTRY **violates** BCNF in STUD, so *decompose* STUD into

  CC, with schema(CC) = {CITY, COUNTRY}, and

  STUD1, with schema(STUD1) = {SNUM, POSTCODE, CITY}
- CC is in BCNF while POSTCODE $\rightarrow$ CITY violates BCNF in STUD1, so *decompose* STUD1 into

  PC, with schema(PC) = {POSTCODE, CITY}, and

  SINFO = {SNUM, POSTCODE}.

Another Example of BCNF Decomposition

Let STUD be a relation schema, with schema(STUD) = {SNUM, POSTCODE, CITY, COUNTRY}, with FDs

{SNUM $\rightarrow$ POSTCODE, POSTCODE $\rightarrow$ CITY, CITY $\rightarrow$ COUNTRY}

- CITY $\rightarrow$ COUNTRY **violates** BCNF in STUD, so *decompose* STUD into

  CC, with schema(CC) = {CITY, COUNTRY}, and

  STUD1, with schema(STUD1) = {SNUM, POSTCODE, CITY}
- CC is in BCNF while POSTCODE $\rightarrow$ CITY violates BCNF in STUD1, so *decompose* STUD1 into

  PC, with schema(PC) = {POSTCODE, CITY}, and

  SINFO = {SNUM, POSTCODE}.
- All the relation schemas in the database schema {CC, PC, SINFO} are now in BCNF

A Third Example

- Consider a modified relation schema EMP, with attributes ENAME, CNAME (child name), DNAME (department name) and MNAME (manager name).
- The set of FDs is F = {E $\rightarrow$ D, D $\rightarrow$ M, M $\rightarrow$ D}, where E stands for ENAME, D stands for DNAME and M stands for MNAME (and C stands for child name).
- All three FDs violate BCNF since EC is the only key.
- We can choose any one of them as the basis for the first decomposition step.
- We will consider all three decompositions in turn.

Third Example: Decomposition 1

- If we first decompose using D $\rightarrow$ M, we get two schemas with attributes {D, M} and {E, C, D}.
- FDs D $\rightarrow$ M and M $\rightarrow$ D are applicable to {D, M}, but both D and M are keys.
- FD E $\rightarrow$ D is applicable to {E, C, D} and E is not a superkey.
- So we decompose {E, C, D} into {E, D} and {E, C}.
- E is a key for {E, D} and EC is the key for {E, C}.
- So the final database schema comprises {D, M}, {E, D} and {E, C}.

Third Example: Decomposition 2

- If we first decompose using E $\rightarrow$ D, we get two schemas with attributes {E, D} and {E, C, M}.
- E $\rightarrow$ D is applicable to {E, D}, but E is a key.
- What FDs are applicable to {E, C, M}?
- None of E $\rightarrow$ D, D $\rightarrow$ M or M $\rightarrow$ D apply because D is not in {E, C, M}.
- *We have to consider all FDs in* F^+.
- Recall that E $\rightarrow$ M follows from E $\rightarrow$ D and D $\rightarrow$ M.
- E $\rightarrow$ M violates BCNF in {E, C, M} because E is not a key.
- So we decompose {E, C, M} into {E, M} and {E, C}.
- So the final database schema comprises {E, D}, {E, M} and {E, C}.

Third Example: Decomposition 3

- If we first decompose using M $\rightarrow$ D, we get two schemas with attributes {M, D} and {E, C, M}.
- FDs D $\rightarrow$ M and M $\rightarrow$ D are applicable to {M, D}, but both D and M are keys.
- Once again we have {E, C, M}, so it is decomposed as before into {E, M} and {E, C}.
- So the final database schema comprises {M, D}, {E, M} and {E, C}.

An example for you to try

Let $\mathcal{R}$ be a relation schema, with schema($\mathcal{R}$) = {C,T,H,R,S,G}.

- C stands for a course,
- T stands for a teacher,
- H stands for hour,
- R stands for room,
- S stands for student and
- G stands for grade.

An example set of FDs $\mathcal{F}$ over $\mathcal{R}$:

1. C $\rightarrow$ T,
2. HR $\rightarrow$ C,
3. HT $\rightarrow$ R,
4. CS $\rightarrow$ G and
5. HS $\rightarrow$ R.

Decompose $\mathcal{R}$ into BCNF.

Dependency Preservation

Recall example: F_3 = {SC $\rightarrow$ P, P $\rightarrow$ C}.
S stands for Street, C stands for City and P stands for Postcode.

{S,C,P} is not in BCNF

Decompose {S,C,P} into {P,C} and {P,S}

P	C
p1	c
p2	c

P	S
p1	s
p2	s

Only FD that can be tested in the decomposition is P $\rightarrow$ C

When we join the two relations, we see that SC $\rightarrow$ P is violated.

Dependency Preservation

A decomposition is **dependency preserving** if the FDs which hold on the original relation schema can be tested on the decomposed schemas, *without using joins*.

We cannot always find a BCNF decomposition that is dependency preserving.

To test that no FDs are violated, we may need to join relations (expensive).

We *can* always find a 3NF dependency-preserving decomposition.

Dependency Preservation

For a starting set of attributes and FDs, some BCNF decompositions may be dependency preserving and some not.

Consider the example with attributes { E, C, D, M } and FDs F = {E $\rightarrow$ D, D $\rightarrow$ M, M $\rightarrow$ D}.

We had three possible decompositions

1. {D, M}, {E, D} and {E, C}.
2. {E, D}, {E, M} and {E, C}.
3. {M, D}, {E, M} and {E, C}.

Which of them is dependency-preserving?

3NF Synthesis Algorithm

Given a relation schema R and a set of FDs F, the following steps produce a 3NF decomposition of R that satisfies the lossless join condition and is dependency preserving:

3NF Synthesis Algorithm

Given a relation schema R and a set of FDs F, the following steps produce a 3NF decomposition of R that satisfies the lossless join condition and is dependency preserving:

1. Find a minimal cover for F, say G.

3NF Synthesis Algorithm

Given a relation schema R and a set of FDs F, the following steps produce a 3NF decomposition of R that satisfies the lossless join condition and is dependency preserving:

1. Find a minimal cover for F, say G.
2. For each FD X $\rightarrow$ A in G, use XA as the schema of one of the relations in the decomposition.

3NF Synthesis Algorithm

Given a relation schema R and a set of FDs F, the following steps produce a 3NF decomposition of R that satisfies the lossless join condition and is dependency preserving:

1. Find a minimal cover for F, say G.
2. For each FD $X \rightarrow A$ in G, use XA as the schema of one of the relations in the decomposition.
3. If none of the schemas from Step 2 includes a superkey for R, add another relation schema that is a key for R.

3NF Synthesis Algorithm

Given a relation schema R and a set of FDs F, the following steps produce a 3NF decomposition of R that satisfies the lossless join condition and is dependency preserving:

1. Find a minimal cover for F, say G.
2. For each FD X $\rightarrow$ A in G, use XA as the schema of one of the relations in the decomposition.
3. If none of the schemas from Step 2 includes a superkey for R, add another relation schema that is a key for R.
4. Delete any of the schemas from Step 2 that is contained in another.

Example of 3NF Synthesis

Recall the example: $F_3 = \{SC \rightarrow P, P \rightarrow C\}$.
S stands for Street, C stands for City and P stands for Postcode.

Example of 3NF Synthesis

Recall the example: F_3 = {SC $\rightarrow$ P, P $\rightarrow$ C}.
S stands for Street, C stands for City and P stands for Postcode.

Step 1 of the algorithm finds that F_3 is a minimal cover.

Example of 3NF Synthesis

Recall the example: F_3 = {SC $\rightarrow$ P, P $\rightarrow$ C}.
S stands for Street, C stands for City and P stands for Postcode.

Step 1 of the algorithm finds that F_3 is a minimal cover.

Step 2 of the algorithm would produce {P,C} and {S,C,P}.

Example of 3NF Synthesis

Recall the example: F_3 = {SC $\rightarrow$ P, P $\rightarrow$ C}.
S stands for Street, C stands for City and P stands for Postcode.

Step 1 of the algorithm finds that F_3 is a minimal cover.

Step 2 of the algorithm would produce {P,C} and {S,C,P}.

Step 3 finds that SC is a superkey.

Example of 3NF Synthesis

Recall the example: F_3 = {SC $\rightarrow$ P, P $\rightarrow$ C}.
S stands for Street, C stands for City and P stands for Postcode.

Step 1 of the algorithm finds that F_3 is a minimal cover.

Step 2 of the algorithm would produce {P,C} and {S,C,P}.

Step 3 finds that SC is a superkey.

Step 4 deletes {P,C} to leave just {S,C,P}.

Another Example

Recall the example: $F_2 = \{E \rightarrow S\}$.
E stands for ENAME, S stands for SAL and C stands for CNAME.

Another Example

Recall the example: $F_2 = \{E \rightarrow S\}$.
E stands for ENAME, S stands for SAL and C stands for CNAME.

Step 1 of the algorithm finds that F_2 is a minimal cover.

Another Example

Recall the example: $F_2 = \{E \rightarrow S\}$.
E stands for ENAME, S stands for SAL and C stands for CNAME.

Step 1 of the algorithm finds that F_2 is a minimal cover.

Step 2 of the algorithm would produce {E,S}.

Another Example

Recall the example: $F_2 = \{E \rightarrow S\}$.
E stands for ENAME, S stands for SAL and C stands for CNAME.

Step 1 of the algorithm finds that F_2 is a minimal cover.

Step 2 of the algorithm would produce {E,S}.

Step 3 finds no superkey, so adds relation schema {E,C}.

Another Example

Recall the example: $F_2 = \{E \rightarrow S\}$.
E stands for ENAME, S stands for SAL and C stands for CNAME.

Step 1 of the algorithm finds that F_2 is a minimal cover.

Step 2 of the algorithm would produce {E,S}.

Step 3 finds no superkey, so adds relation schema {E,C}.

Step 4 finds nothing to delete.

A Third Example

Consider the example: $F = \{AB \rightarrow CD, C \rightarrow AD, D \rightarrow A\}$.

A Third Example

Consider the example: $F = \{AB \rightarrow CD, C \rightarrow AD, D \rightarrow A\}$.

Step 1 of the algorithm finds that F is **not** a minimal cover.

A Third Example

Consider the example: $F = \{AB \rightarrow CD, C \rightarrow AD, D \rightarrow A\}$.

Step 1 of the algorithm finds that F is **not** a minimal cover.

First we form a canonical set of FDs:
$\{AB \rightarrow C, AB \rightarrow D, C \rightarrow A, C \rightarrow D, D \rightarrow A\}$.

A Third Example

Consider the example: $F = \{AB \rightarrow CD, C \rightarrow AD, D \rightarrow A\}$.

Step 1 of the algorithm finds that F is **not** a minimal cover.

First we form a canonical set of FDs:
$\{AB \rightarrow C, AB \rightarrow D, C \rightarrow A, C \rightarrow D, D \rightarrow A\}$.

Then we find that $AB \rightarrow D$ and $C \rightarrow A$ are redundant.

A Third Example

Consider the example: $F = \{AB \rightarrow CD, C \rightarrow AD, D \rightarrow A\}$.

Step 1 of the algorithm finds that F is **not** a minimal cover.

First we form a canonical set of FDs:
$\{AB \rightarrow C, AB \rightarrow D, C \rightarrow A, C \rightarrow D, D \rightarrow A\}$.

Then we find that $AB \rightarrow D$ and $C \rightarrow A$ are redundant.

So we are left with minimal cover
$G = \{AB \rightarrow C, C \rightarrow D, D \rightarrow A\}$.

A Third Example

Consider the example: $F = \{AB \rightarrow CD, C \rightarrow AD, D \rightarrow A\}$.

Step 1 of the algorithm finds that F is **not** a minimal cover.

First we form a canonical set of FDs:
$\{AB \rightarrow C, AB \rightarrow D, C \rightarrow A, C \rightarrow D, D \rightarrow A\}$.

Then we find that $AB \rightarrow D$ and $C \rightarrow A$ are redundant.

So we are left with minimal cover
$G = \{AB \rightarrow C, C \rightarrow D, D \rightarrow A\}$.

The rest is easy.

An example for you to try

Consider the set of attributes { Drinker, Address, Pub, Location, Beer, Cost }, along with the following set of FDs:

- Drinker $\rightarrow$ Address
- Pub $\rightarrow$ Location
- Pub, Beer $\rightarrow$ Cost, Location

Produce a set of 3NF relation schemas for the above.