Functional dependencies

- Given a schema $R(T)$ and $X\,Y \subseteq T$,
  a FD is a constraint on the instances of R.
  $X \to Y$ (X functionally determines Y) iff
  $\forall r$ VALID instance of R
  $\forall t_1, t_2 \in r$
  if $t_1[X] = t_2[X]$ then $t_1[Y] = t_2[Y]$

NB Using De Morgan we can say
$\exists t_1, t_2 \in r$ if $t_1[X] = t_2[X]$ AND $t_1[Y] \neq t_2[Y]$
$\Rightarrow \forall r$ NOT a valid instance of R | ? Not sure

NB Whenever you introduce a variable,
always say if it $\exists$ or $\forall$

- What does it mean for a specific instance to satisfy a FD ?

Given an instance r_0 of R
it is said to satisfy the FD $X \to Y$ ($r_0 \models X \to Y$)
if the property holds for r_0

(in formula) $r_0 \models X \to Y$ iff $\forall t_1, t_2 \in r_0$, $t_1[X] = t_2[X] \Rightarrow t_1[Y] = t_2[Y]$

Expressing FD

'BookCode → Title' is an indication of a BAD schema

Consider BookCode → Author

1. direct expr. ⇝ $BC = \Rightarrow A =$
   "if BC is the same, then A is also same"

2. contraposition ⇝ $A \neq \Rightarrow B \neq$
   "if A different then BC different"

$A \Rightarrow B$
is the same as
$\neg A \Rightarrow \neg A$

3. absurd ⇝
   "it is impossible to have two lines with the same BC and different A"

? ⇝ $\neg(BC = \wedge A \neq)$
⇝ $BC = \wedge A \neq \Rightarrow False$

$A \Rightarrow B$
$\neg A \cup B$
$\neg(A \wedge \neg B)$
$(A \wedge \neg B) \Rightarrow False$

nothing can imply false so this is the way to express impossibility in logic

the RULE

$True \wedge A \wedge B \Rightarrow C \vee D \vee False$

$True \wedge A \wedge \neg D \Rightarrow C \vee \neg B \vee False$

DeMorgan

$\neg(A \wedge B) \equiv \neg A \vee \neg B$

$\neg(A \cup B) \equiv \neg \;\; \wedge \neg B$

Examples of FD

1) At any given time, a teacher is at most in a classroom

$\rightsquigarrow$ time= $\wedge$ teacher= $\Rightarrow$ room=

(also) It is impossible to have the same teacher at the same time in different classrooms

$\rightsquigarrow$ time= $\wedge$ teacher= $\wedge$ room $\neq$ $\Rightarrow$ false

(then we set all equals to write the FD)

2) It is not possible for two different teachers to be in the same classroom at the same time

$\rightsquigarrow$ teacher $\neq$ $\wedge$ room= $\wedge$ time= $\Rightarrow$ False

room= $\wedge$ time= $\Rightarrow$ teacher=

3) If two lessons take place on different floors, they belong to two different degree courses

$\rightsquigarrow$ lesson $\neq$ $\wedge$ floor $\neq$ $\Rightarrow$ degree $\neq$

$\rightsquigarrow$ lesson $\neq$ $\wedge$ floor $\neq$ $\wedge$ degree= $\Rightarrow$ false

$\rightsquigarrow$ degree $\Rightarrow$ lesson= $\vee$ floor=

(correct solution is)

Degree= $\Rightarrow$ floor=

4) If two different lessons take place on the same day for the same subject, they belong to two different CDL

$\rightsquigarrow$ lesson $\neq$ $\wedge$ day= $\wedge$ subject= $\Rightarrow$ degree $\neq$

(imp.) $\rightsquigarrow$ lesson $\neq$ $\wedge$ day= $\wedge$ subject= $\wedge$ degree= $\Rightarrow$ false

(FD) $\rightsquigarrow$ day= $\wedge$ subject= $\wedge$ degree= $\Rightarrow$ lesson=

FD Notation

- $R < T, F >$
  denotes a schema with attributes T and f.d. F

- a FD is COMPLETE (bad name for "left side minimal")
  when the FD does not hold if we remove attributes

  ⇝ $X \to Y$ is complete $\Leftrightarrow \forall W \subset X$, $X \to Y$ is NOT valid

  (example) fiscal_code, ~~surname~~ → name   NOT complete because fc already has name

- a set of attribute X is a SUPERKEY (superset of a key $X \subseteq X$)
  when X determines every other attribute of the relation $X \to T$

  (example) student_id → T
  fiscal-code → T
  student_id, fiscal-code → T

  given T as 'every other attribute', all these set of attributes T are superkeys

- an attribute is a KEY if it is a SUPERKEY and it is COMPLETE,
  meaning that $X \to T$ and is left side minimal

More examples on FD

1) When two rooms are in a different floor,
then they have a different number of seats

(wrong) ⇝ room ≠ ∧ floor ≠ ⇒ seats ⇝ room ≠ is redundant because it is already in floor ≠

(NEVER BE REDUNDANT)

(correct) ⇝ floor ≠ ⇒ seats ≠

(imp) ⇝ floor ≠ ∧ seats = ⇒ False

2) It is not possible to have the same teacher
that teaches the same subject in two different DegreeCourses

(imp) ⇝ teacher = ∧ subject = ∧ degree ≠ ⇒ false

(FD) ⇝ teacher = ∧ subject ⇒ degree

FD implications

There are FDs that implies other FDs.
So fixed a set of FDs other FDs are implicated by F.

(def) $r \models F \Rightarrow r \models X \to Y \quad \forall r$ ← instance of relation R

↑ set of FDs

⇝ What does it mean that
F logically implies $X \to Y$ (written $F \models X \to Y$)?
It means that for every instance of relation r ($\forall r$),
if r satisfies F ($r \models F$) then $r \models X \to Y$

- But we cannot prove that because we would need to check $\forall r$

Example

Let r be an instance of $R\langle T, F \rangle$, with $F = \{X \to Y, X \to Z\}$
and $X, Y, Z \subseteq T$. Let $X' \subseteq X$.
There are other FDs satisfied by R, such as

- (trivial FD) $X \to X'$
- $X \to Y, Z$ ⇝ because $X \to Y$ and $X \to Z$ so X determines both Y, Z
  so $\{X \to Y, X \to Z\} \models X \to YZ$
- $\{X \to Y, Y \to Z\} \models X \to Z$

Inference rules (or derivation rules)

These are the building blocks of problems.

We use a set of rules that are correct and complete.

Armstrong's "axioms" (actually "rules")

1. if $Y \subseteq X$, then $X \to Y$ (reflexivity R)
2. if $X \to Y$, $Z \subseteq T$, then $XZ \to YZ$ (augmentation A) → whenever you have an implication, you can add the same set left and right
3. if $X \to Y$, $Y \to Z$, then $X \to Z$ (transitivity T)

Derivation (or mechanical deduction)

(def) Let F be a set of FD,
$X \to Y$ is derivable from F ($F \vdash X \to Y$),
if $X \to Y$ can be inferred from F using Armstrong's axioms

We say $F \vdash X \to Y$ means that starting from F
we can derive the FD $X \to Y$.

Applying these rules, we can make our set of dependencies F
bigger and bigger.

Example

$R(A, B, C, D)$

$F: \{A \to B, BC \to D\}$

AC is a superkey? In other words, $AC \to ABCD$?

⇝ Remember, a superkey is a set of attributes X that determines every other attribute.

It is very difficult to reason, so we use the axioms (or deduction rules).

Let's start from $A \to B$ that is the first FD

using augmentation $AC \to BC$ because $C \subseteq T$ where T is every other attribute

because then I want to exploit the second FD $BC \to D$.

Now, we augment also the second FD $BC \to D$ with Aug(BC)

that becomes $BC \to BCD$ because BC set union BC is just BC.

We know that $AC \to BC$ and $BC \to BCD$,

so using transitivity we get $AC \to BCD$.

In the end, we augment with A and get $AC \to ABCD$.

We proved that AC is a superkey without any reasoning, just with inference rules.

Correctness and Completedness of Armstrong's axioms

(theorem) Armstrong's axioms are correct and complete

The fact that by using these I will discover true FDs is obvious.

1. Correctness (easy)

   $\forall F, f \quad F \vdash f \Rightarrow F \models f$

   $\vdash$ derives using rules; $\models$ implies

   a set of FDs F can be used to derive with axioms $\vdash$ / imply by reasoning $\models$ another set of FDs f

2. Completeness (difficult and more important)

   $\forall F, f \quad F \models f \Rightarrow F \vdash f$

   It means that everything that can be proven can be discovered also by applying deduction rules only.

(3. Decidability ?)

We have an algorithm that can answer in polynomial time, if f can or cannot be derived from F

Closure of a set of dependencies F

Closure is a set operator that has a property such that $A \subseteq A^+ = (A^+)^+$.
It means take a set, make it bigger until you arrive
to some form of boundary where you cannot make it bigger anymore.

(def) Given a set F of FD,
the closure of F, denoted by F^+, is

$F^+ = \{X \to Y \mid F \vdash X \to Y\}$ ⇝ all the dependencies $X \to Y$ that you can deduce by inference from F

For example.
$A \to B$ in a relation $R(A, B, C)$
the closure of $A \to B$, that is $\{A \to B\}^+$
contains things like $A \to A$, $B \to B$, $AB \to A$, $ABC \to B$,
$AC \to B$,

We are writing everything that can be deduced from $A \to B$ and $R(A, B, C)$.
First, we write all the trivial FD, like

$ABC \to AB$	$AC \to A$	$AB \to B$	$BC \to BC$	
$ABC \to AC$	$AC \to C$	$AB \to A$	$BC \to B$	...
$ABC \to BC$	$AC \to AC$	$AB \to AB$	$BC \to C$	

This is done using the rule of reflexivity,
which is $Y \subseteq X \vdash X \to Y \quad \forall F$
that is whenever X is a subset of Y, $X \to Y$ can be deduced from F.
Even starting from the empty set $\{\}^+ = \{X \to Y \mid Y \subseteq X\}$

(continuation of closures of FDs)

Then, we have all the FDs that we get by augmentation and by Transitivities.

This is very interesting because we can say that

$F \vdash X \to Y \iff X \to Y \in F^+$ ⇝ F determines $X \to Y$ if and only if $X \to Y$ belongs to the closure of F

However, computing the closure of F is expensive because grows exponentially.
So, it is useless in practice.
And we will use a new definition.

Closure of a set of attributes X

(or X^+ if F is clear from context)

(def) Given $R<T, F>$ and $X \subseteq T$,
the closure of X with respect to F (denoted by X_F^+) is

$$X_F^+ = \{A_i \in T \mid F \vdash X \to A_i\}$$

⤳ an attribute A_i belongs to this set T if from X you can deduce that X determines A_i

For example.

$$F = \{CF \to SID,\ SID \to N\}$$

$$CF_F^+ = \{SID,\ N\}$$

obvious $CF \to SID$

this is because in the closure of CF I can find also the name

So, the closure of an attribute are all those attributes that depend on this attribute directly or applying any of the Armstrong's Axioms.

Fundamental theorem

$$X \to Y \in F^+ \iff F \vdash X \to Y \iff Y \subseteq X_F^+$$

this was already stated in the previous pages

⤳ Y is included in the closure of X

This means that if closing X I reach all attributes in Y, then X determines Y.

1. $X \to Y \in F^+$ ⤳ this is exponential
2. $F \vdash X \to Y$ ⤳ we don't know how to solve it (or try to apply all axioms randomly)
3. $Y \subseteq X_F^+$ ⤳ this can be done in polynomial time

Algorithm for computing the closures of X

Let's see an example.

$F = \{DB \to E, B \to C, A \to B\}$. Compute $(AD)^+$

⇝ $X^+ = AD$ initialize the result variable with attributes A and D.

Then scan repeatedly the FDs in F.

Remember that $X_F^+ = \{A_i \in T \mid F \vdash X \to A_i\}$

Assume that two lines are equal on A and D, look at FDs, what can you reach? Given $A \to B$, we can reach $X^+ = ADB$.

Now we restart the scan with ADB and see what we can reach, that is $X^+ = ADBE$, given $DB \to E$ by transitivity.

The same happens with $B \to C$, and we get $X^+ = ADBEC$.

We try once more to check that there's nothing left.

We observe that this algorithm is polynomial, the max number of scans $\leq$ number of attributes a

$\leq$ number of dependencies p

meaning that max scans $\leq \min(a, p)$.

The cost is given by $p \cdot a \cdot \min(a, p) \approx n^3$, so it is polynomial based on size of the input.

Certainly it is not exponential.

There is another algo, called 'fast closures' that is n^2, but we are not studying it.

Slow Closure (pseudocode)

input $R\langle T, F\rangle$, $X \subseteq T$

output X^+

algo $\rightsquigarrow$ $X^+ = X$

    while X^+ changes

        for $W \to V$ in F with $W \subseteq X^+$ and $V \not\subseteq X^+$

            $X^+ = X^+ \cup$

The question is:

→ "is $X \to Y$ derivable from F $(F \vdash X \to Y)$?"

just close X and see if it does reach all attributes in Y $(Y \subseteq X^+_F)$

→ "is X a superkey?

just compute the closure and see if you can reach all other attributes.

→ "is X a key? (minimal superkey)

check if there is a subset of X that is still a key, eliminating one by one all the attributes in the set and computing the closures.

Observe that this is still polynomial.

Finding a key

$F = \{DB \to E,\ B \to C,\ A \to B\}$ compute $(AD)^+$

find a key.

⇝ start from a superkey,
like the set of all attributes, that is always a superkey
because it trivially determines itself.

Let's start with $\cancel{A}BCDE$

$(BCDE)^+ = BCDE$ because you cannot reach A from BCDE since A is not in the left side of my FD, so we need it

Try with $A\cancel{B}CDE$

$(ACDE)^+ = ACDEB$ we can reach B from $A \to B$, so it is a superkey

Now $A\cancel{C}DE$, after removing B

$(ADE)^+ = ADEBC$ because of $A \to B$ and $B \to C$, so superkey

Now $A\cancel{D}E$

$(AE)^+ = AEBC$ but we cannot reach D because it is never on the right side

And $AD\cancel{E}$

$(AD)^+ = ADBCE$ so AD is a key!

This algo is guaranteed to always find at least a key.
But in the worst case the complexity is $n!$
(exponential complexity)

Prime Attributes

An attribute is prime when it belongs to at least one key.
The problem of checking if an attribute is prime is NP-complete (exponential)

(def) Given a schema $R<T,F>$,
we say that $W \subseteq T$ is a candidate key of R if

- $W \to T \in F^+$ ⇝ W superkey
- $\forall V \subset W,\ V \to T \notin F^+$ ⇝ if $V \subset W$, V not a superkey

Equivalence of FD. sets

It is possible to present the same set of dependencies in many similar ways.

$$F = \left\{ \begin{array}{l} FC \to SID \\ SID \to FC \\ FC \to N, S, A \end{array} \right\} \qquad G = \left\{ \begin{array}{l} FC \to SID \\ SID \to FC \\ SID \to N, S, A \end{array} \right\}$$

They are equivalent!

Observe that $F \vdash G \iff G \subseteq F^+$

and $G \vdash G \iff F \subseteq G^+$

(def) Two sets of FDs, F and G, are equivalent $F \equiv G$ on the scheme R, iff $F^+ = G^+$

So, from now on we focus on F^+, that is the reality (the complete truth) while F is just one way to tell the story.

Choosing the best set of FDs (or cover)

And what if I want to make F as small as possible, going in the other direction with respect to closures.

$$\left\{ \begin{array}{l} FC, N \to S \\ FC \to N \end{array} \right\} \equiv \left\{ \begin{array}{l} FC \to N \\ FC \to S \end{array} \right\}$$

The smaller, the better. We prefer the one on the right.

When is a set of FDs minimal?

A cover is minimal when

1. there are no extraneous attributes that you may delete (like N before)
2. no redundant FDs that you may delete

An attribute A is extraneous when $F \vdash (X - \{A\}) \to Y$, meaning that it may be deleted and X still determines Y

A FD $X \to Y$ is redundant when $F - \{X \to Y\} \vdash X \to Y$ meaning that it can be derived from F

Canonical Covers

A set of dependencies F is called canonical iff

1. the right part of FD in F is an attribute.

$$ABC \to DEF \quad \rightsquigarrow \quad \begin{array}{l} ABC \to D \\ ABC \to E \\ ABC \to F \end{array}$$

2. there are no extraneous attributes
3. no dependency in F is redundant

There is an algo to get the canonical form of F

1. transform FD in the form $X \to A$
2. delete extraneous attributes
   $\rightsquigarrow$ for example, given $ABC \to E$ how do I know if A is extraneous?
   meaning $F \vdash \cancel{A}BC \to E$? compute BC_F^+ and check if $E \in BC_F^+$
3. eliminate redundant dependencies
   $\rightsquigarrow$ take $F - (AB \to C)$ and compute $(AB)^+$
   is $C \in AB^+$?

The canonical cover is not necessarily unique.

Exercise

Consider $R(AB \to D,\ AD \to B,\ A \to C,\ C \to A,\ BEC \to D,\ ABCDE)$

1) Are there any extraneous attributes in some dependencies?

⇝ A is extraneous when $F \vdash (X - \{A\}) \to Y$,
meaning that if we delete A, we still can get Y.
Compute $(X - \{A\})^+$ and check if $Y \in (X-\{A\})^+$.
(Do it with all FDs?)

1. $\cancel{A}B$ ⇝ $B^+ = B$ $\quad A\cancel{B}$ ⇝ $A^+ = AC$ but $D \notin A^+$
2. $\cancel{A}D$ ⇝ $D^+ = D$ $\quad A\cancel{D}$ ⇝ $A^+ = AC$ but $B \notin A^+$
3. $\cancel{B}EC$ ⇝ $EC^+ = EC$ $\quad B\cancel{E}C$ ⇝ $BC^+ =$ ~~BCA~~ but ~~$D \notin BC^+$~~

No extraneous attributes

ghelli said
$BC^+ = BCAD$
so $D \in BC^+$
maybe because
$B\cancel{E}C \to D$

2) Are there any redundant dependencies?

⇝ $X \to Y$ is redundant when $F - \{X \to Y\} \vdash X \to Y$,
meaning that it can be derived from F

Take $F - (AB \to C)$ and see if $C \in AB^+$ ← we compute X^+ on $F - (X \to Y)$

Let's try eliminating FDs one by one and see if I can still reach the same attributes

1. $F - (AB \to D)$ ⇝ $AB^+ = ABCD$ because we have $B\cancel{E}C \to D$, so it is REDUNDANT
2. $F - (AD \to B)$ ⇝ $AD^+ = ADC$ NOT reaching B
3. $F - (A \to C)$ ⇝ $A^+ = A$ NOT reaching C
4. $F - (C \to A)$ ⇝ $C^+ = C$
5. $F - (BEC \to D)$ ⇝ $BEC^+ = BECAD$ ← we can reach everything, in particular D, so this is REDUNDANT

but if we remove $AB \to D$ then we cannot remove this because otherwise we cannot reach D anymore

Consider $R(AB \to D,\ AD \to B,\ A \to C,\ C \to A,\ BEC \to D,\ ABCDE)$

3. How many keys can you find?

A key is basically a left side minimal superkey.

Let's start finding the superkeys, meaning the attributes X that determine every other attribute.

We start from the set of all attributes that is the trivial superkey.

$(ABCDE)^+ = ABCDE$ because of reflexivity axiom. $\left(\begin{array}{l}\text{if } Y \subseteq X \\ \text{then } X \to Y\end{array}\right)$

Now we try to eliminate attributes one by one.

$\cancel{A} \rightsquigarrow (BCDE)^+ = BCDEA$ superkey

$\cancel{AB} \rightsquigarrow (CDE)^+ = CDEAB$ superkey and also a key!

$\cancel{ABC} \rightsquigarrow (DE)^+ = DE$ Not a superkey

$\cancel{ABD} \rightsquigarrow (CE)^+ = CEA$ not a superkey

$\cancel{ABE} \rightsquigarrow (CD)^+ = CDAB$ not a superkey, because E is never on the right side of any FD, so it must be in every key to be reachable

$A\cancel{B}CDE \rightsquigarrow (ACDE)^+ = ACDEB$ superkey

$A\cancel{B}\cancel{C}DE \rightsquigarrow (ADE)^+ = ADECB$ superkey

$A\cancel{B}\cancel{C}\cancel{D}E \rightsquigarrow (AE)^+ = AEC$ NOT a superkey

Observe that we can find the key both on the original or the simplified set because they are equivalent.

Decomposition of schemas

- As we know, to resolve redundancy in schemes, we decompose into smaller equivalent schemes.
  It is a decomposition because the SET UNION of the new attributes is the set of original attributes

(def) Given a schema $R(T)$,

$\rho = \{R1(T1), \dots, Rk(Tk)\}$ is a decomposition of R

iff $\bigcup T_i = T$

- How do we choose the right decomposition?
  We do not want to loose information (lossy decomposition)
  In a lossless decomposition, we can get the original table with a natural join.
  Use projection to create smaller tables.

What if there are already existing procedures, queries?
We just create a view of the original table from new tables.

- Lossless join decompositions (very important definition)

(def) $\rho = \{R1(T1), \dots, R_K(T_K)\}$

is a lossless join decomposition of $R(T)$

iff for each VALID instance r of R

$r = (\pi_{T1}\, r) \bowtie \dots \bowtie (\pi_{TK}\, r)$ ⤳ doing projections and join we get the original table

Observe that this must be valid also for all the values that I will insert in the future.
We need a theorem.

It is easy to prove that $r \subseteq (\pi_{T1}\, r) \bowtie \dots \bowtie (\pi_{TK}\, r)$ ⤳ the way we loose information is by getting a bigger table with spurious tuples

Binary decompositions

← theorem of Binary Decomposition

There is a nice theorem which states that whenever there is a binary decomposition the decomposition is lossless

iff the intersection of the two schemas is a superkey either for T_1 and T_2, meaning

$$(T_1 \cap T_2) \rightarrow T_1 \in F^+ \quad \text{or} \quad (T_1 \cap T_2) \rightarrow T_2 \in F^+$$

It is essential that the column being used for connecting two tables is a superkey at least for one table.
This is a necessary and sufficient condition.
If your attribute is not a superkey for at lest one table, then the decomposition is lossy.

How to check? We compute the closures contains all the attributes of T_1 or T_2 then it's okay.

The next step is to find one decomposition, not only check.

Projection of FDs

This is necessary to go beyond binary decomposition.

(def) Given the schema $R\langle T, F\rangle$ and $T1 \subseteq T$,
the projection of F over T1 is
$$\pi_{T_1}(F) = \{ X \to Y \in F^+ \mid X, Y \subseteq T_1 \}$$

Basically we want to project original FDs to the table where they belong to after decomposition.
First, consider their closures (also called meanings).
Second, if X, Y belongs to T1, then they are projected into T_1.

Let's see an example.
Let $R(A,B,C)$ and $F = \{A \to B, B \to C, C \to A\}$

$R1(AB) \rightsquigarrow \pi_{AB}(F) \equiv \{A \to B,\ B \to A\}$

$B \to A$ is from $B \to C$ and $C \to A$ (closures)

$R2(AC) \rightsquigarrow \pi_{AC}(F) \equiv \{A \to C,\ C \to A\}$

The steps are
1. compute the closures
2. syntactical projection
3. reduce to canonical form

synthetic projection means eliminating every dependency which is about attributes that are not in the table

remove redundant dependencies, like the trivial ones

How to calculate $\pi_{T_1}(F)$?
Calculate the determinants of T_1, meaning all the subsets of T_1 $(Y \subseteq T_1)$.
Then for each det, compute the closures.
And in the end you remove redundancy.

$A \to \cancel{A}B$
$B \to A\cancel{B}$
~~$AB \to AB$~~

Preserving dependencies

(def) Given the schema $R<T, F>$, the decomposition $\rho = \{R_1, \dots, R_n\}$ preserves dependencies if the union of dependencies in $\pi_i(F)$ is a cover of F.

for example.

$R(ABC, \{B \to C, C \to A\})$

$R_1(AB)$ $\{B \to A\}$ $\quad$ $R2(BC)$ $\{B \to C\}$

Is $C \to A$ lost? Yes, because in the decomposition $C^+ = C$

This can happen but it is not a problem.

It is a desirable property, but still not crucial.

(theorem) If there is an ennary decomposition that preserves dependencies and such that at least one T_j is a superkey, the ρ is a lossless join decomposition.

This is a generalization of the binary decomp. theorem.

It is sufficient but not necessary.

Normal Forms

There are different types.

- 1NF ⇝ each attribute has an elementary type (like number, date, string, ...)
  - ↳ in NFNF (Not First Normal Form) also called object-relational system, where in a single cell you can put a list, array, ...
- 2NF, 3NF ⇝ impose restrictions on dependencies
- BCNF (Boys Codd Normal Form) ⇝ it is the most natural and restrictive and the one we are studying

BCNF

Intuition: if there exists in R a non-trivial $X \to A$ dependency and X is not key, then X models the identity of an entity other than those modeled by by the whole R

In a table, given $X \to A$, if X is a superkey in that specific table, then there is no redundancy.
While, if X is not a superkey, then we have redundancy.

(def) $R<T, F>$ is in BCNF $\iff$ for every $X \to A \in F^+$ with $A \notin X$ (meaning X is non-trivial) and X is a superkey

So $X \to A$ must be trivial or, if not trivial, X must be a superkey.
It means that you cannot have two different lines with the same X.
Observe that we can express it in two alternative ways

1. $A \in X$ or X superkey ⇝ True $\Rightarrow$ $A \in X$ or X superkey
2. $A \notin X \Rightarrow$ X superkey

Since F^+ is exponential, we cannot directly verify this property.
However, there is a theorem that we can use.
It states that if the BC property is true for every FD in F,
it is also true for every FD in F^+.
Meaning that we can check in F^+ and that's enough.

(theorem) $R<T,F>$ is in BCNF $\iff$ for every $X \to A \in F$ with $A \notin X$ (non trivial), X is a superkey

It is the same as the definition, just with F instead of F^+.

Why the definition is with F^+?
Because it is a property of F^+, depends on the meaning, not on a cover F.
But if we check a cover, we know that the property will be valid for all the covers.

Not clear...

If F is in canonical form, then each FD does not contain any extraneous attribute nor any redundant FD,
so each superkey is also a key.
Meaning that in canonical cover we just check keys.

Observe that in the definition we have $X \to A$
that is an atomical FD because A is an attribute,
while X is a set of attributes.

Example

1. Teachers (TaxCode, Name, Dep, DepAddress)

   ⤳ We have a redundancy because dep → dep, depAddress meaning that each professor will have a depAddress which is repeated for those of the same dep.

   This is not in BCNF.

2. Employees (Code, Qualification, ChildName)

   ⤳ Given code → Qualification, if an employee has 3 children it will appear 3 times.

   But $(code)^+$ = qualification, so it is not a superkey, because we need also the childname.

   This is not in BCNF

3. Libraries (BookCode, ShopName, ShopAddress, Title, Quantity)

   ⤳ Given BookCode → Title
   ShopName → ShopAddress
   BookCode, ShopName → Quantity ← this is the only superkey

   But since the first two are FDs not trivial and not with superkeys, this cover will be redundant.

4. Telephones (AreaCode, Number, City, Subscriber, Street) ← not mobile phones

   ⤳ Given $F = \{AC, NU \rightarrow CI, SU, ST$
   $CI \rightarrow AC\}$

   But the city CI alone is not a superkey.
   This is not BCNF.

The Analysis Algorithm (also the splitting algorithm)

Consider a schema where some FD violate the BCNF and display it.

Let's see an example.

Teachers (TaxCode, Name, Dep, DepAddress)

$F = \{$ TaxCode $\rightarrow$ Name, Dep

Dep $\rightarrow$ DepAddress $\}$

The algo analyse each FD to see if it is or not in BCNF.

$(\text{TaxCode})^+ =$ TaxCode, Name, Dep, DepAddress $\rightsquigarrow$ this is a key

$(\text{Dep})^+ =$ Dep, DepAddress $\rightsquigarrow$ NOT a key

How to solve it? We create a table with the closure of the attribute that violates the BCNF, so $(\text{Dep})^+$, and another table with all the other attributes + X itself.

(algo) For every $X \rightarrow Y$ that violates BCNF,

$R\langle T, F \rangle$ is recursively decomposed in

$R1(X^+)$ and $R2(X, T - X^+)$

foreign key

So R1 (Dep, DepAddress) and R2 (Dep*, TaxCode, Name)

$\downarrow$ Dep $\rightarrow$ DepAddress

$\downarrow$ TaxCode $\rightarrow$ Name, Dep

Now when we project the dependencies and Dep is a superkey of its table.

This is a Binary Decomposition and it is guaranteed to be lossless because of the theorem we have seen.

The algorithm guarantees BCNF and lossless decomposition, but does not guarantee the FDs preservation.

Steps of the algo

1. choose the FD that violates BCNF
2. split the schema
3. project the FD on the new tables
4. apply recursively

Example.

Telephones (AreaCode, Number, City, Subscriber, Street)

$$F = \left\{ \begin{array}{l} AC, NU \to CI, SU, ST \\ \quad CI \to AC \end{array} \right\} \begin{array}{l} \rightsquigarrow AC, NU \text{ is a superkey} \\ \rightsquigarrow CI \text{ is } \underline{\text{not}} \text{ a superkey} \end{array}$$

We split based on CI.

$R1(\underline{CI}, AC) \qquad R2(\underline{NU}, CI^*, SU, ST)$

We project the FD.

$R1(F) = \{CI \to AC\} \qquad R2(F) = \{\}$

syntactic projection means take all the π whose attributes are included in R2

Observe that if we calculate only the syntactic projection R2 has no dependencies because they all contain AC which is not present in R2.

But we should compute all the closures, included with 2,3,.. terms

$CI^+ = \ldots \quad , \quad NU^+ = \ldots , \ldots$

$(CI, NU)^+ = \cancel{CI}, \cancel{NU}, \cancel{AC}, SU, ST$ and eliminating the trivial one,

we now can say $R2(F) = \{CI, NU \to SU, ST\}$

Observe that $AC, NU \to CI$ has been lost, but it is not important.

3NF

It is weaker than BCNF.

(def) $R<T, F>$ is in 3FN if for every $X \to A \in F^+$, with $A \notin X$, X is a superkey or A is prime

(theo) $\leadsto$ same as (def) but with F instead of F^+

Remember that A is prime when it belongs to at least one key.
Note that A is on the right side of the FD.

Since the BCNF might loose some FDs,
we might decide to accept the 3NF with some redundancy
to keep all the FDs.

To sum up, the 3NF admits a non-trivial and non-key dependency if the attributes on the right are prime;
while BCNF never admits any non-trivial and non-key dependence.

The Synthesis Algorithm (to reach 3NF)

It groups attributes together, in the opposite direction with respect to the analysis.
It guarantees the lossless join property, the preservation of FDs,
but not the BCNF.
However, it might happen that BCNF is reached.

Let's see an example

$R(ABCDE, \{AD \to E,\ E \to B,\ E \to D,\ C \to E\}$

First, group all the dependencies that have the same left side

$G1 = \{AD \to E, AD \to B\}$. $G2 = \{E \to D\}$ $G3 = \{C \to E\}$

Then for every group define a relation

$R1(ADEB)$ $R2(ED)$ $R3(CE)$

Third, delete all relations that are included in others.

$R1(ADEB)$ $R3(CE)$ $\leadsto$ we deleted R2 because $R2 \subseteq R1$

Finally, check if one of these relations is a superkey.
This is because of the theorem that says
$\leadsto$ whenever you have a decomposition which preserves any FDs
if one of the relations of the decomposition is a superkey
of the original table,
then you are guaranteed to have a lossless join.

How can we be sure that no FDs are lost?
Because of the relation step.

Let's check the superkeys. $R1+ = ADEB$ $R3+ = CED$ No superkeys

We compute a key.

$\cancel{A}BCDE+ = BCDE$ $\leadsto$ we need A, C

$ACDE+ = \dots$ $\leadsto$ B not needed

$ACE+ = ACEDB$

$AC+ = ACEDB$ $\leadsto$ it is a key (and the only one)

So, $R1(ADEB)$, $R3(CE)$, $R4(AC)$

Steps of the Synthesis Algo

Let $R<T, F>$ with F canonical cover

1. partition into groups based on the determinant
2. define a relationship for each group with the same determinant
3. if a schema is contained into another, delete the smaller
4. check that at least one schema built is a superkey, otherwise find one arbitrary key and add that schema

Provided that you start with a set of FDs that are in canonical form, this algo is guaranteed to give a schema in 3NF.

Which algo is better?
It depends on the goal you want to reach.

Which one is more computationally expensive?
Synthesis is polynomial, only the key is expensive (if needed).
Analysis is more difficult, because the projection of new tables is computationally expensive.

Deciding if a schema is in BCNF, it is polynomial, while deciding if it is in 3NF it is NP-hard.

Example of Analysis Algo

$R(ABCDE, \{AD \to E, AD \to B, E \to D, C \to E\})$

Scan all the FDs and check whether they violate the BC condition, by computing closures.

$AD^+ = ADEB$ not a superkey

We draw a tree because the operation is recursive, and the result will be in the leaves of the tree.

Now we compute the syntactic projection and if no FD is lost, then because of a theorem, the syntactic projection coincide with the real projection.

If a FD is on one side, it cannot be on the other.

R

$R1(ADEB)$ $\quad$ $R2(ADC)$ $\quad$ $(AD) \cup (T - AD^+)$

$\{AD \to E,\ AD \to B,\ E \to D\}$ $\quad$ $\{\}$

We observe from the tree that we have lost $C \to E$, so we need to compute the real projection.

It is extremely expensive, compute the closure of every subset of R1 with respect to R

$A^+ = A$	$AD^+ = ADEB$	ADE^+
$D^+ = D$	AE^+	ADB^+
$E^+ = ED$	AB^+	AEB^+
$B^+ = B$	DE^+	DEB^+
$\rightsquigarrow$	DB^+	
no new FDs	EB^+	

We basically look for the closure where we get a new FD because we used $C \to E$, that is the lost FD.

But we will never arrive anywhere because C is never on the right side.

In this case, the two projection coincide with the syntactical projection.

$R(ABCDE, \{AD \to E, AD \to B, E \to D, C \to E\})$

Let's try now on the right side, with $R2(ADC)\ \{\}$

$A+ = A$ $\quad$ $AD+ = ADBE$

$D+ = D$ $\quad$ $AC+ = ACD$

$C+ = CED$ $\quad$ $DC+ = DC$

this is interesting because, while
$C \to C$ is trivial and $C \to E$ is useless because
we do no not have E in $R2$,
$C \to D$ can be projected in $R2$,
meaning that we have learnt a new FD,
since it was not in the syntactal projection.

R

$R1(ADEB)$ $\quad$ $R2(ADC)$

$\{AD \to E,\ AD \to B,\ E \to D\}$ $\quad$ $\{C \to D\}$

TEST | Here you are authorized during test
to skip the step of finding real projections,
just mention it.

Now. Is $R1$ in BCNF? Let's compute the closures to check superkeys.
$AD+ = ADEB$ superkey, $E+ = ED$ ~~NOT~~ NOT superkey, then split

$R1(ADEB)$

$(E) \cup (\Gamma - E+)$

$R3(ED)$ $\quad$ $R4(EAB)$

$\{E \to D\}$ $\quad$ $\{\}$ ← should compute projections...

$R2(ADC)$

$R5(CD)$ $\quad$ $R6(CA)$

$\{C \to D\}$ $\quad$ $\{\}$

The leaves are in BCNF.
But we have lost a lot of FDs.