Relational DB Design Theory

CSC 343

Winter 2021

MICHAEL LIUT (MICHAEL.LIUT@UTORONTO.CA)
ILIR DEMA (ILIR. DEMA@UTORONTO.CA)
DEPARTMENT OF MATHEMATICAL AND COMPUTATIONAL SCIENCES
UNIVERSITY OF TORONTO MISSISSAUGA

UNIVERSITY OF TORONTO MISSISSAUGA

Introduction

- There are always many different schemas for a given set of data.

  e.g. you could combine or divide tables.

- How do you pick a schema? Which is better? What does “better” mean?

- Fortunately, there are some principles to guide us.

Schemas and Constraints

- Consider the following sets of schemas:

  Students(utorid, name, email)

  vs.

  Students(utorid, name)

  Emails(utorid, address)

- Consider also:

  House(street, city, value, owner, propertyTax)

  vs.

  House(street, city, value, owner)

  TaxRates(city, value, propertyTax)

Constraints are domain-dependent

Avoid Redundancy

This table has redundant data, and that can lead to anomalies.

name	addr	beersLiked	manf	favBeer
Janeway	Voyager	Bud	A.B.	WickedAle
Janeway	Voyager	WickedAle	Pete's	WickedAle
Spock	Enterprise	Bud	A.B.	Bud

Update anomaly: if Janeway is transferred to *Intrepid*, will we remember to change each of her tuples?

Deletion anomaly: if nobody likes Bud, we lose track of the fact that Anheuser-Busch manufactures Bud.

Database Design Theory

- Allows us to improve a schema systematically.
- The general idea is to:
  1. express constraints on data; and
  2. use these to decompose the relations.
- Ultimately, get a schema that is in ***normal form***.
  - “Normal” meaning conforming to a standard.
  - “Normal Form” referring to guaranteeing ‘good’ properties; such as no anomalies.
- The process of converting a schema to a normal form is called ***normalization***.

Part I

Functional Dependency Theory

Functional Dependencies

UNIVERSITY OF TORONTO MISSISSAUGA

- Let's say "**X → Y** holds in R", this means that "**X** functionally determines **Y**".

- Conventions:

  - …, X, Y, Z represent sets of attributes; A, B, C,… represent single attributes.
  - No braces used for sets of attributes, just ABC, rather than {A,B,C}.

- Why ***functional dependency***?
  - "functional" because there is a mathematical function that takes a value for **X** and gives a unique value for **Y**.
  - "dependency" because the value of **Y** depends on the value of **X**.

Functional Dependencies (FDs)

- Need a special type of constraint to help us with normalization.

X → Y is an assertion about relation R that whenever two tuples of R agree on all the attributes in set X, they must also agree on all attributes in set Y.

e.g. Let's say that X = {AB} and Y = {C}

R

A	B	C
x1	y1	c2
x1	y1	c2
x2	y2	c3

Properties about FDs

1. Rules
   - Trivial FDs
   - Splitting/Combining
   - Armstrong's Axioms

2. Algorithms related to FDs
   - Closure (of a set of attributes of a relation)
   - Minimal Basis (of a relation)

Rule: Trivial FDs

- Not all functional dependencies are useful.
  - A → A always holds
  - ABC → A also always holds.

The right side is a subset of the left side.

- A functional dependency with an attribute on both sides.
  - ABC → AD becomes ABC → D

    OR
  - Delete the trivial FDs:

    ABC → A and ABC → D becomes just ABC → D

This is called "singleton form".

Rule: Splitting/Combining

$X \rightarrow A_1A_2 \ldots A_n$ holds for R exactly when each of $X \rightarrow A_1$, $X \rightarrow A_2$, …, $X \rightarrow A_n$ hold for R.

e.g. $A \rightarrow BC$ is equivalent to $A \rightarrow B$ and $A \rightarrow C$

e.g. $A \rightarrow F$ and $A \rightarrow G$ is equivalent to $A \rightarrow FG$

- There is no splitting rule for the left side.

e.g. $ABC \rightarrow DEF$ is NOT equivalent to $AB \rightarrow DEF$ and $C \rightarrow DEF$

- Usually, FDs are expressed with singleton right sides.

Example: FDs

Drinkers(name, addr, beersLiked, manf, favBeer)

Reasonable FDs to assert:

name → addr, favBeer

Note this FD is the same as: name → addr and name → favBeer

beersLiked → manf

Example: FDs

name	addr	beersLiked	manf	favBeer
Janeway	Voyager	Bud	A.B.	WickedAle
Janeway	Voyager	WickedAle	Pete's	WickedAle
Spock	Enterprise	Bud	A.B.	Bud

name → addr

beersLiked → manf

name → favBeer

Rule: Armstrong's Axioms

X, Y, Z are sets of attributes

1. **Reflexivity**: if $X \supseteq Y$, then $X \rightarrow Y$.

2. **Augmentation**: if $X \rightarrow Y$, then $XZ \rightarrow YZ$ for any Z.

3. **Transitivity**: if $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$.
4. **Union**: if $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$.
5. **Decomposition**: if $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$.

Transitive Property

The transitive property holds for FDs

- Consider the FDs: A → B and B → C; then A → C holds.

- Consider the FDs: AD → B and B → CD; then AD → CD holds or just AD → C (because of trivial FDs).

How do you identify FDs?

- FDs are based on domain knowledge
  - Intrinsic features of the data that is specific to your use case.
  - Something you know (or assume) about the data.

- Database engines cannot identify FDs for you
  - Designer must specify them as part of the schema.
  - DBMS can only enforce FDs when told to do so.

- DBMSs cannot "optimize" FDs
  - It has only a finite sample of the data.
  - A FD constrains the entire domain.

FDs are a Generalization of Keys

- Superkey: X → R
  - A superkey must include all of the attributes of the relation on the right-hand side (RHS).

- A Functional Dependency: X → Y
  - A FD can involve just a subset of the key.

e.g. House (street, city, value, owner, tax)

street, city → value, owner, tax *(both FD and key)*

city, value → tax *(FD only)*

Inferring FDs

- Given a set of FDs, it is often possible to infer further FDs.

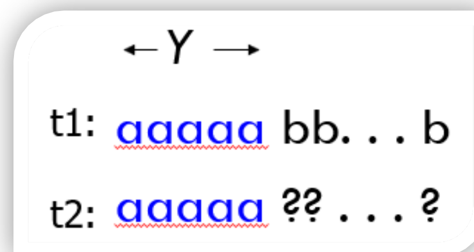
- Let's assume that we have the following FDs:

$$X_1 \rightarrow A_1,$$

$$X_2 \rightarrow A_2,$$

$$\ldots,$$

$$X_n \rightarrow A_n.$$



- Does the FD $Y \rightarrow B$ also hold in any relation that satisfies the given FDs?
  - To prove this, you must assume that two tuples agree on all attributes of Y

Example: Inferring FDs

For Example:

if $A \rightarrow B$ and $B \rightarrow C$ holds, then surely $A \rightarrow C$ holds,

even if we do not explicitly say so.

$A \rightarrow C$ is ***entailed (implied)*** by $\{A \rightarrow B, B \rightarrow C\}$

Example: Inferring FDs in General

UNIVERSITY OF TORONTO MISSISSAUGA

ClientID	Income	OtherProd	Rate	Country	City	State
225	High	A	2.1%	USA	San Francisco	MD
420	High	A	2.1%	USA	San Francisco	CA
333	High	B	3.0%	USA	San Francisco	CA
576	High	B	3.0%	USA	San Francisco	CA
128	Low	C	4.5%	UK	Reading	Berkshire
193	Low	C	4.5%	UK	London	London
550	Low	B	3.5%	UK	London	London

F1: [Income, OtherProd] → [Rate]

F2: [Country, City] → [State]

How to prove it in the general case?

Algorithm: Closure Test

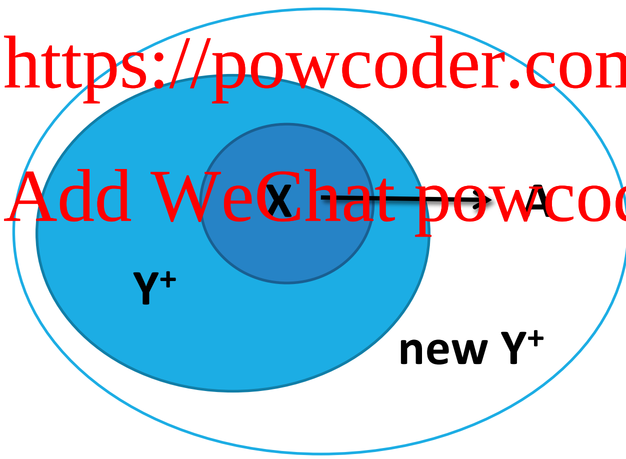
- Closure Test for Functional Dependencies
- Given attribute set Y and FD set F:

  Denote Y_F^+ (or Y^+) the closure of Y relative to F

  Y_F^+ is the set of all FDs given or implied by Y
- Computing the closure of Y
  1. Start with: $Y_F^+ = Y$, $F' = F$
  2. While there exists an $f \in F'$ s.t. $LHS(f) \subseteq Y_F^+$:
     - $Y_F^+ = Y_F^+ \cup RHS(f)$
     - $F' = F' - f$
  3. Stop when: $Y \rightarrow B$ for all $B \; Y_F^+$

Algorithm: Closure

- Computing the closure Y^+ given attribute set Y and FD set F:

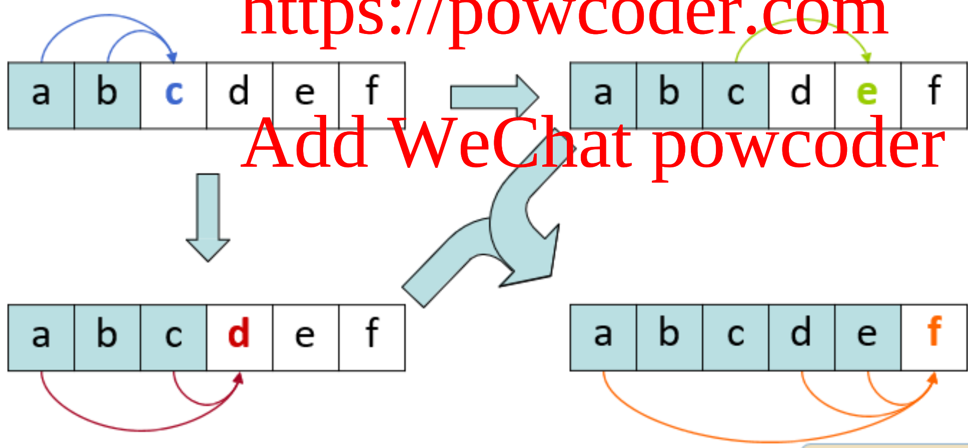


Example: Closure

Consider R(a,b,c,d,e,f)

with FDs: ab -> c, ac -> d, c -> e, ade -> f

Find Y^+ if Y = ab or find $\{a,b\}^+$



{a,b}⁺={a,b,c,d,e,f} or *ab -> cdef*

ab is a candidate key!

Example: Closure

- Given:

  F: $AB \rightarrow C$
  $A \rightarrow D$
  $D \rightarrow E$
  $AC \rightarrow B$

X	X_F^+
A	$\{A, D, E\}$
AB	$\{A, B, C, D, E\}$
AC	$\{A, C, B, D, E\}$
B	$\{B\}$
D	$\{D, E\}$

- Question:

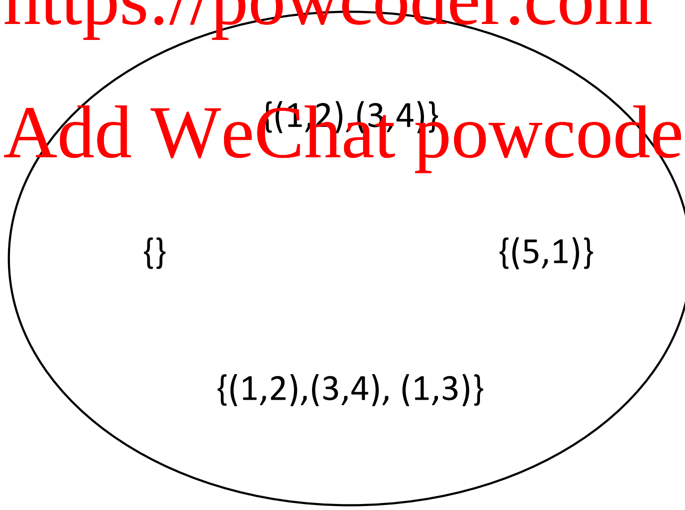
1. Is AB → E entailed by ***F***? ***YES***
2. Is D → C entailed by ***F***? ***NO***

- Result: X_F^+ allows us to determine all FDs of the form X → Y entailed by ***F***.

A Geometric View of FDs

- Imagine the set of all ***instances*** of a particular relation.
- That is, all finite sets of tuples have the proper number of components.
- Each instance is a point in space.

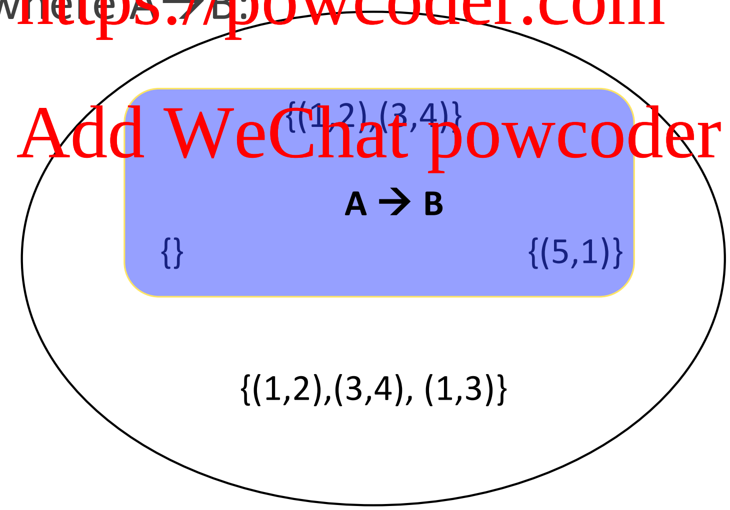
e.g. R(A,B):



A FD is a Subset of Instances

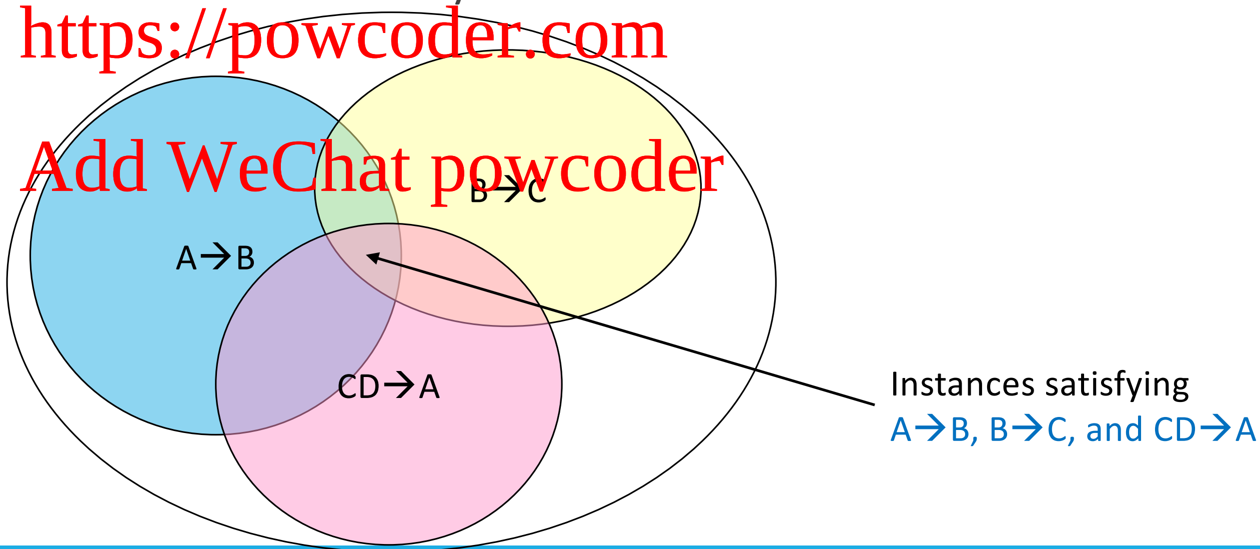
- For each $X \rightarrow A$ there is a subset of all instances that satisfy the FD.
- Therefore, FDs can be represented by a region in the space.

e.g. R(A,B), where $A \rightarrow B$:



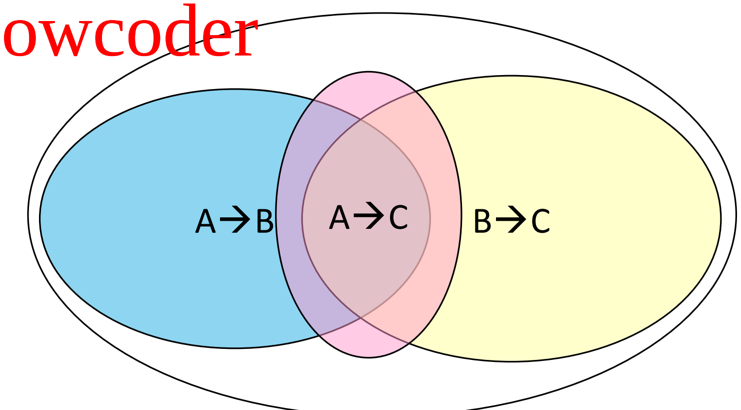
Representing Sets of FDs

- If each FD is a set of relation instances, then a collection of FDs correspond to the ***intersection*** of those sets.
  - intersection: all instances that satisfy all of the FDs.



Implication of FDs

- If a FD $Y \rightarrow B$ follows from the FDs $X_1 \rightarrow A_1, \ldots, X_n \rightarrow A_n$, then the region in space of instances for $Y \rightarrow B$ must include the intersection of the regions for the FDs $X_i \rightarrow A_i$.
  - Every instance satisfying all the FDs $X_i \rightarrow A_i$ surely satisfies $Y \rightarrow B$.
  - However, an instance could satisfy $Y \rightarrow B$, yet not be in this intersection.
- For a set of FDs ***F***, ***F***$^+$ is the set of all FDs implied by ***F***.



Part II

Schema Decomposition

Relational Schema Design

Drinkers(name, addr, beersLiked, manf, favBeer)

- Goal is to avoid redundancy and the anomalies it enables.
  - ***Update anomaly***: one occurrence of a fact is changed, but not all occurrences have been updated.
  - ***Deletion anomaly***: valid fact is lost when a tuple is deleted.

Recall the FDs: name→addr,favBeer and beersLiked→manf.

name	addr	beersLiked	manf	favBeer
Janeway	Voyager	Bud	A.B.	WickedAle
Janeway	???	WickedAle	Pete's	???
Spock	Enterprise	Bud	???	Bud

Data is redundant, because the ??? can be figured out by using the FDs.

Goal of Decomposition

- Eliminate redundancy by decomposing a relation into several relations.
- Check that a decomposition does not lead to "bad design".

❖ You want to identify a "good" method to split relations.
  - ✓ Splitting a relation into 2+ smaller relations, limiting redundancy.
  - ✓ Splitting F into subsets which apply to the new relations.
    - ✓ Compute the projection of functional dependencies to check!

Schema Decomposition

Given relation R and FDs F

- Split R into R_i s.t. for all i, $R_i \subset R$ (no new attributes)
- Split F into F_i s.t. for all i, F entails F_i (no new FDs)
- F_i involves only attributes in R_i

Caveat: entirely possible to lose information

- F^+ may entail FD f which is not in $(U_i\ F_i)^+$

=> Decomposition lost some FDs

- Possible to have $R \subset \bowtie_i R_i$

=> Decomposition lost some relationships

Goal: minimize anomalies without losing info

An issue with decomposition is ***information loss*** – we may not be able to reconstruct the corresponding instance of the original relation.

Good Properties of Decomposition

1. Lossless Join Decomposition
   - When we join decomposed relations we should get **exactly** what we started with.

2. Avoid Anomalies
   - Avoid redundant data.

3. Dependency Preservation
   - $(F_1 \cup \ldots \cup F_n)^{+} = F^{+}$

Example: Splitting Relations

Student Name	Student Email	Course	Instructor
Alice	alice@utoronto.ca	CSC 343	Liut
Alice	alice@utoronto.ca	CSC 209	Petersen
Bob	bob@utoronto.ca	CSC 148	Zingaro
Laura	laura@utoronto.ca	CSC 343	Dema

Students (email, name)

Courses (code, instructor)

Taking (email, courseCode)

Students $\bowtie$ Taking $\bowtie$ Courses has additional tuples!

- (Alice, alice@utoronto.ca, CSC 343, Jones), but Alice is not in Dema's section of CSC 343
- (Laura, laura@utoronto.ca, CSC 343, Liut), but Laura is not in Liut's section of CSC 343

Why did this happen? How to prevent it?

Information Loss with Decomposition

- Decompose R into S and T
  - Consider the FD $A \rightarrow B$, with A only in S and B only in T.

- FD Loss
  - Attributes A and B are no longer in the same relation; you must join T and S to enforce $A \rightarrow B$ (which is expensive!).
- Join Loss
  - Neither $(S \cap T) \rightarrow S$ nor $(S \cap T) \rightarrow T$ in F^+
    - Joining T and S will produce extraneous tuples.

Property: Lossless Join Decomposition

- Often confused with "Lossy Decomposition".
  - Lost of information is unavoidable when retrieving the initial relation.

- A decomposition should not lose information!

- A decomposition ($\mathbf{R}_1, \ldots, \mathbf{R}_n$) of a schema, **R**, is **lossless** if every valid instance, r, of **R** can be reconstructed from its components:

$$r = r_1 \bowtie \ldots \bowtie r_n \text{ where } r_i = \Pi_{R_i}(r)$$

Example: Lossless Decomposition

r

ID	Name	Addr
11	Pat	1 Main
12	Jen	2 Pine
13	Jen	3 Oak

$r_1 = \Pi_{R1}(r)$

ID	Name
11	Pat
12	Jen
13	Jen

$r_2 = \Pi_{R2}(r)$

Name	Addr
Pat	1 Main
Jen	2 Pine
Jen	3 Oak

$r_1 \bowtie r_2$

ID	Name	Addr
11	Pat	1 Main
12	Jen	2 Pine
13	Jen	3 Oak
12	Jen	3 Oak
13	Jen	2 Pine

- Loses the fact that (12, Jen) lives at 2 Pine (not 3 Oak)
- Loses the fact that (13, Jen) lives at 3 Oak

Remember: lossy decompositions yield ***more*** tuples than they should when relations are joined together.

Testing for Losslessness

- A (binary) decomposition of **R** = (R, **F**) into **R**$_1$ = (R_1, **F**$_1$) and **R**$_2$ = (R_2, **F**$_2$) is lossless if and only if:
  - either the FD $(R_1 \cap R_2) \rightarrow R_1$ is in **F**$^+$; <u>**OR**</u>
  - the FD $(R_1 \cap R_2) \rightarrow R_2$ is in **F**$^+$.
  - all attributes common to both R_1 and R_2 functionally determine ALL the attributes in R1; <u>**OR**</u>
  - all attributes common to both R_1 and R_2 functionally determine ALL the attributes in R_2.

Example: Decomposition Property

In our example
- *Name* $\not\rightarrow$ *ID,Name*
- *Name* $\rightarrow$ *Name, Addr*

A lossless decomposition
- *[ID,Name] and [ID,Addr]*

Example 2:
- *Category* $\rightarrow$ ModelName, Category
- *Category* $\not\rightarrow$ Price, Category
- Better to use [MN, Category] and [MN, Price]

In other words, if R1 ∩ R2 forms a superkey of either R1 or R2, the decomposition of R is a lossless decomposition

Example: Lossless Decomposition

- A decomposition is lossless if we can recover:

R(A, B, C)

Decompose

R1(A, B)   R2(A, C)

Recover

R'(A, B, C)

Thus,   R' = R

Example: Lossless Decomposition

- Given:
  - *Lending-schema* = (*branch-name, branch-city, assets, customer-name, loan-number, amount*)
  - FDs: *branch-name* → *branch-city, assets* and *loan-number* → *amount, branch-name*

- Decompose Lending-schema into two schemas:
  - *Branch-schema* = (*branch-name, branch-city, assets*)
  - *Loan-info-schema* = (*branch-name, customer-name, loan-number, amount*)

Show that the decomposition is a Lossless Decomposition

Example: Lossless Decomposition

- Decompose Lending-schema into two schemas:
  - *Branch-schema* = (*branch-name, branch-city, assets*)
  - *Loan-info-schema* = (*branch-name, customer-name, loan-number, amount*)

Since *Branch-schema* ∩ *Loan-info-schema* = {*branch-name*}

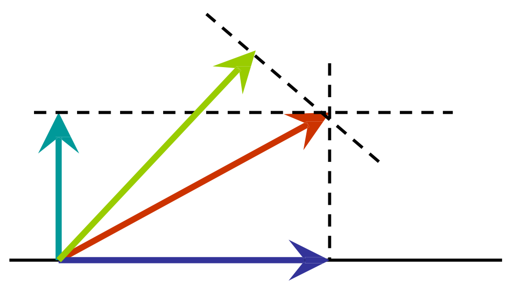
We are given: *branch-name* → *branch-city, assets*

Therefore, this is a lossless decomposition.

Projecting FDs

- Once we have split a relation, we have to re-factor our FDs to match.
  - Each FD must only mention attributes from one relation.

- Similar to geometric projection.
  - Many possible projections (depends on how we slice it).
  - Keep only the ones we need (minimal basis).



Part III
Normal Forms

Motivation for Normal Forms

- To assist us in identifying a "good" schema
  - Everyone may have a slightly different definition of "good", but there are some guidelines that assist us in achieving what we all want (avoiding anomalies, reducing/eliminating redundant information, etc…).

- Many Normal Forms:
  - 1st
  - 2nd
  - 3rd
  - Boyce-Codd
  - … and many others which we won't discuss …

BCNF $\subseteq$ *3NF* $\subseteq$ *2NF* $\subseteq$ *1NF*

1^{st} Normal Form (1NF)

- No multi-valued attributes allowed.
  - Imagine storing a list of values in an attribute.
- Counterexample
  - Course(name, instructor, [student,email]*)

Name	Instructor	Student Name	Student Email
CSC 343	Liut	Alice	alice@utoronto.ca
CSC 343	Liut	Alice	alice@utoronto.ca
CSC 343	Liut	Bob	bob@utoronto.ca
CSC 148	Simion	Laura	laura@utoronto.ca

2nd Normal Form (2NF)

- Non-prime attributes depend on candidate keys.
  - Consider non-prime attributes A
  - Then there exists a FD X s.t. X → A, and X is a candidate key.

- Counterexample
  - Movies(title, year, star, studio, studioAddress, salary)
  - FDs: title, year → studio and studio → studioAddress and star → salary

Title	Year	Star	Studio	StudioAddr	Salary
Star Wars	1977	Hamill	LucasFilm	1 Lucas Way	$100,000
Star Wars	1977	Ford	LucasFilm	1 Lucas Way	$100,000
Star Wars	1977	Fisher	LucasFilm	1 Lucas Way	$100,000
Patriot Games	1992	Ford	Paramount	Cloud 9	$2,000,000
Last Crusade	1989	Ford	LucasFilm	1 Lucas Way	$1,000,000

3rd Normal Form (3NF)

- Non-prime attributes depend ***only*** on candidate keys.
  - Consider FD X → A
  - Either X is a superkey OR A is ***prime*** (part of a key)
- Counterexample
  - studio → studioAddr (studioAddr depends on studio which is not a candidate key)

Title	Year	Studio	StudioAddr
Star Wars	1977	LucasFilm	1 Lucas Way
Patriot Games	1992	Paramount	Cloud 9
Last Crusade	1989	LucasFilm	1 Lucas Way

3NF, Dependencies, and Join Loss

Theorem: always possible to convert a schema to become lossless join and dependency-preserving.

Caveat: always possible to create schemas in 3NF for which these properties do not hold.

- FD Loss Example 1:
  - MovieInfo(title, year, studioName)
  - StudioAddress(title, year, studioAddress)

  Cannot enforce studioName → studioAddress

- Join Loss Example 2:
  - Movies(title, year, star)
  - StarSalary(star, salary)

  Movies ⋈ StarSalary yields additional tuples

Boyce-Codd Normal Form (BCNF)

- One additional restriction over 3NF
  - All non-trivial FDs have superkey LHS

- Counterexample
  - CanadianAddress(street, city, province, postalCode)

  - Candidate keys: {street, postalCode}, {street, city, province}
  - FD: postalCode → city, province

  - Satisfies 3NF: city, province both prime
  - Violates BCNF: postalCode is not a superkey

Possible anomalies involving postalCode

Boyce-Codd Normal Form (BCNF)

- A relation R is in BCNF if, whenever, X → A is a non-trivial FD that holds in R, X is a superkey.
  - Recall: non-trivial means that A is not contained in X.

- A relation not in BCNF: Drinkers(name, addr, beersLiked, manf, favBeer) with the FDs name → addr, favBeer and beersLiked → manf.
  - The only key is: {name, beersLiked}
  - In each FD, the left side is **not** a superkey.
  - Any one of these FDs shows *Drinkers* is not in BCNF.

Example: BCNF

Beers(name, manf, manfAddr) FDs: name→manf and manf→manfAddr

- ❑ Beers w.r.t. name→manf does not violate BCNF, but manf→manfAddr does.

- ❖ BCNF required that the only FDs that hold are the result of key(s).

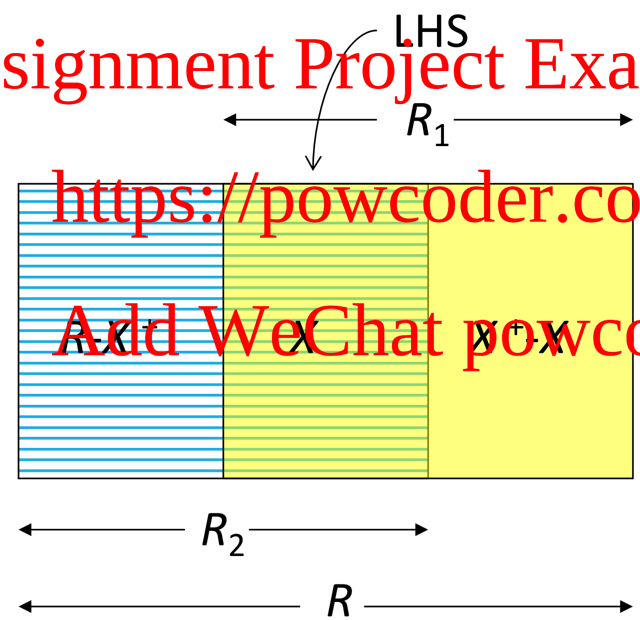
UNIVERSITY OF TORONTO MISSISSAUGA

Decomposition into BCNF

- Given: relation R with FDs F
- Look among the given FDs for a BCNF violation $X \rightarrow Y$ (i.e., X is not a superkey)

- Compute X^+.
  - Find $X^+ \neq X \neq$ all attributes, (otherwise X is a superkey)
- Replace R by relations with:
  - $R_1 = X^+$.
  - $R_2 = R - (X^+ - X) = R - X^+ \cup X$
- Continue to recursively decompose the two new relations
- *Project* given FDs F onto the two new relations.

Decomposition on X → Y



What do we want from a decomposition?

- ✓ *Lossless Join* : it should be possible to project the original relations onto the decomposed schema, and then reconstruct the original,

i.e. retrieve the original tuples

- ✓ *No anomalies*

- ✓ *Dependency Preservation* : All the original FDs should be satisfied.

BCNF Decomposition

- What do we get from a BCNF decomposition?

  - *Lossless Join* : ✓

  - *No anomalies* : ✓
  - *Dependency Preservation* : ✗

3NF Decomposition

- What do we get from a 3NF decomposition?

  - *Lossless Join* : ✓
  - *No anomalies* : ✗
  - *Dependency Preservation* : ✓

Unfortunately, neither BCNF nor 3NF can guarantee all three properties we want.

Limits of Decomposition

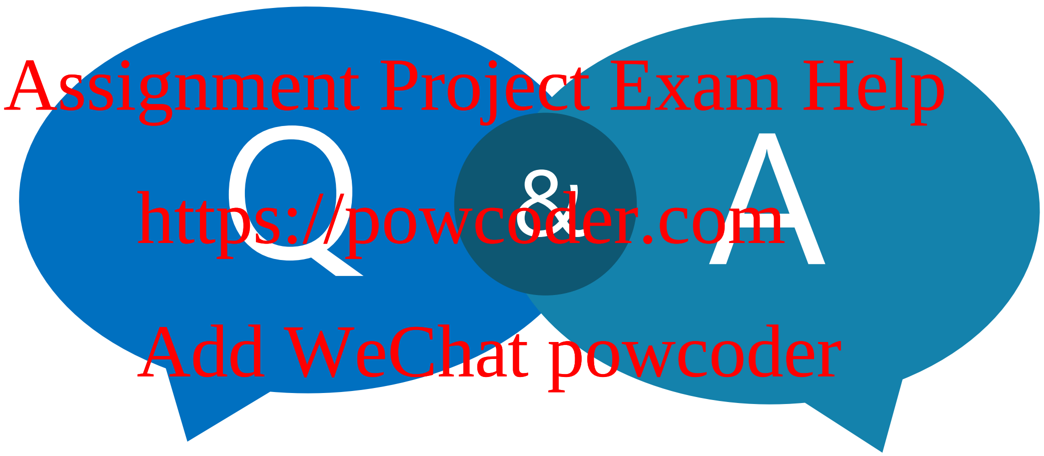
- Pick two…
  - Lossless Join
  - Dependency Preserving
  - Anomaly-Free

- 3NF
  - Provides lossless join and dependency preservation.
  - May allow anomalies.

- BCNF
  - Anomaly-free and lossless join.
  - Sacrifice dependency preservation.

Use domain knowledge to choose 3NF vs. BCNF

Questions?



THANKS FOR LISTENING

I'LL BE ANSWERING QUESTIONS NOW

UNIVERSITY OF TORONTO MISSISSAUGA

Citations, Images and Resources

Database Management Systems (3rd Ed.), Ramakrishnan & Gehrke

Some content is based off the slides of Dr. Fei Chiang - http://www.cas.mcmaster.ca/~fchiang/

Supporting Content

Class Exercise!

- You will be given 5 minutes to prove if *$AB \rightarrow F$* holds in relation R(A, B, C, D, E, F), given the FDs:

AB → C,

BC → AD,

D → E,

CF → B.

- *Hint*: you are computing the closure!

Algorithm:

1. Start with: $Y_F^+ = Y$, $F' = F$
2. While there exists an $f \in F'$ s.t. $LHS(f) \subseteq Y_F^+$:
   - $Y_F^+ = Y_F^+ \cup RHS(f)$
   - $F' = F' - f$
3. Stop when: $Y \rightarrow B$ for all B Y_F^+

Solution: Class Exercise!

- Iterations:
  - X = {A,B} Use: AB→C
  - X = {A,B,C} Use: BC→AD
  - X = {A,B,C,D} Use: D→E
  - X = {A,B,C,D,E} No more changes to X are possible

- The FD: CF→B cannot be used because its left side is never contained in X.

Algorithm: Minimal Basis

- A.k.a.: Minimal Cover. Slightly different from Canonical Cover.
  - A canonical cover allows more than one attribute on the RHS.

- Minimal Bases is the opposite of closure.
- Given a set of FDs ***F***, find the minimal ***F'*** s.t. F' ∈ F and F' entails f for all f ∈ F'

- Properties of a Minimal Basis F' are:
  - RHS is always singleton
  - If any FD is removed from F', F' is no longer a minimal basis.
  - If for any FD in F' we remove one or more attributes from the LHS of f ∈ F', the result is no longer a minimal basis.

Algorithm: Minimal Basis

- Minimal Basis for functional dependencies:
  - Right sides are singleton.
  - No FD can be removed.
  - No attribute can be removed from a left side.

- Constructing a Minimal Cover:
  - Decompose the RHS to single attributes.
  - Repeatedly try to remove an attribute from a LHS and see if the removed attribute can be derived from the remaining FDs.
  - Repeatedly try to remove a FD and see if the remaining FDs are equivalent to the original set (i.e. does the closure of the LHS attributes, with removed FD, include the RHS attribute?).

Algorithm: Minimal Basis

- Formally, it is straightforward but time-consuming;

1. Split all RHS into singletons
2. For all f in F', test whether J = $(F' – f)^+$ is still equivalent to F^+, as it might make F' too small.

3. For all i ∈ LHS(f), for all f ∈ F', let LHS(f') = LHS(f) – i

   ** test whether $(F' – f + f')^+$ is still equivalent to F^+ **

4. Repeat (2) and (3) until neither makes progress.

Example: Minimal Basis

Given a relation R(A, B, C, D) and a defined set of FDs F = {A→AC, B→ABC, D→ABC}, find the minimal basis ***M*** of F.

1st Step: H = {~~A→A~~, A→C, B→A, ~~B→B~~, ~~B→C~~, ~~D→A~~, D->B, ~~D→C~~}

2nd Step:

- A->A, B->B: can be removed as trivial
- A->C: can't be removed, as there is no other LHS with A
- B->A: can't be removed, because for J=H-{B→A} is B^+=BC
- B->C: can be removed, because for J=H-{B→C} is B^+=ABC
- D->A: can be removed, because for J=H-{D→A} is D^+=DBA
- D->B: can't be removed, because for J=H-{D→B} is D^+=DC
- D->C: can be removed, because for J=H-{D→C} is D^+=DBAC

Step 2 outcome: H = {A→C, B→A, D→B}

Example: Minimal Basis

3^{rd} Step

– H does not change as all LHS in H are single attributes.

4^{th} Step

– H does not change.

Minimal Basis: ***M*** = H = {A→C, B→A, D→B}

Finding *Minimal Basis* can be complicated! You will work through examples in tutorial!

Algorithm: Projecting FDs

Projecting FDs

Given: a relation R, with a set of FDs F that hold in R, and a relation $R_i \subset R$.
Determine: the set of all FDs F_i that they follow from F and involve only attributes of R_i.

FD Projection Algorithm

1. Start with $F_i = \emptyset$
2. For each subset X of R_i:
    a. Compute X^+.
    b. For each attribute A in X^+: if A is in R_i, then add X → A to F_i.
3. Compute the minimal basis of F_i.

Improving Projection's Efficiency

- Ignore trivial dependencies.
  - There is no need to add X → A if A is in X itself.

Even with these tricks, projection is still expensive!

- Ignore trivial subsets.
  - The empty set or the set of all attributes (both are subsets of X).

- Ignore supersets of X if $X^+ = R$.
  - They can only give us "weaker" FDs (with more on the LHS).

Example: Projecting FDs

- ABC with FDs $A \rightarrow B$ and $B \rightarrow C$
  - $A^+ = ABC$; yields $A \rightarrow B$, $A \rightarrow C$
    - We ignore A→A as it is trivial.
    - We ignore the supersets of A, AB^+ and AC^+, because they can only give us "weaker" FDs (with more on the LHS).
  - $B^+ = BC$; yields $B \rightarrow C$
  - $C^+ = C$; yields nothing.
  - $BC^+ = BC$; yields nothing.

- Resulting FDs: $A \rightarrow B$, $A \rightarrow C$, and $B \rightarrow C$
- Projection onto AC : $A \rightarrow C$
  - Only FD that involves a subset of {A,C}
- Projection on BC: $B \rightarrow C$
  - Only FD that involves subset of {B,C}

Example: BCNF
Failure to Preserve Dependencies

UNIVERSITY OF TORONTO MISSISSAUGA

- Suppose we start with $R(A,B,C)$ and FDs $AB \rightarrow C$ and $C \rightarrow B$

- There are two keys: $\{A,B\}$ and $\{A,C\}$.

- $C \rightarrow B$ is a BCNF violation, so we must decompose it into AC, BC.

The problem is that if we use *AC* and *BC* as our database schema, we cannot enforce the FD $AB \rightarrow C$ in these decomposed relations.

3NF avoids this problem

- *3rd Normal Form* (3NF) modifies the BCNF condition so we do not have to decompose in this problem situation.

- An attribute is *prime* if it is a member of any key.

- $X \rightarrow A$ violates 3NF if and only if X is not a superkey, and also A is not prime.

i.e. it's ok if X is not a superkey, as long as A is prime.

Example: 3NF Preserves Dependencies

- In our problem situation with FDs $AB \rightarrow C$ and $C \rightarrow B$, we have keys AB and AC.

- Thus, A, B, and C are each prime.

- Although $C \rightarrow B$ violates BCNF, it does not violate 3NF.

Algorithm: 3NF Synthesis

- We can always construct a decomposition into 3NF relations with a lossless join and dependency preservation.
- Need *minimal basis* for the FDs (same as used in projection)
  - Right sides are single attributes.
  - No FD can be removed.
  - No attribute can be removed from a left side.
- One relation for each FD in the minimal basis.
  - Schema is the union of the left and right sides.
- If no key is contained in an FD, then add one relation whose schema is some key.

Example: 3NF Synthesis

- Relation R(ABCD) with FDs A → B and A → C.
- Decomposition: AB and AC from the FDs, with AD for a key.