Design Theory for Relational DBs: Functional Dependencies, Decompositions, Normal Forms

Introduction to databases
CSCC43 Winter 2012
Ryan Johnson

Thanks to Manos Papagelis, John Mylopoulos, Arnold Rosenbloom and Renee Miller for material in these slides

Database Design Theory

• Guides systematic improvements to database schemas
• General idea:
  – Express constraints on the data
  – Use these to decompose the relations
• Ultimately, get a schema that is in a “normal form” that guarantees certain desirable properties
• “Normal” in the sense of conforming to a standard
• The process of converting a schema to a normal form is called normalization

Goal #1: remove redundancy

• Consider this schema

<table>
<thead>
<tr>
<th>Student Name</th>
<th>Student Email</th>
<th>Course</th>
<th>Instructor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xiao</td>
<td>xiao@gmail</td>
<td>CSCC43</td>
<td>Johnson</td>
</tr>
<tr>
<td>Xiao</td>
<td>xiao@gmail</td>
<td>CSCD08</td>
<td>Bretscher</td>
</tr>
<tr>
<td>Jaspreet</td>
<td>jaspreet@utsc</td>
<td>CSCC43</td>
<td>Johnson</td>
</tr>
</tbody>
</table>

• What if...
  – Xiao changes email addresses? (update anomaly)
  – Xiao drops CSCD08? (deletion anomaly)
  – UTSC creates a new course, CSCC44 (insertion anomaly)

Multiple relations => exponentially worse

Goal #2: expressing 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)

Dependencies, constraints are domain-dependent
Part I: Functional Dependencies

Functional dependencies

- Let \( X, Y \) be sets of attributes from relation \( R \)
- \( X \rightarrow Y \) is an assertion about tuples in \( R \)
  - Any tuples which agree in all attributes of \( X \) must also agree in all attributes of \( Y \)
- "\( X \) functionally determines \( Y \)"
  - Or, "The values of attributes \( Y \) are a function of those in \( X \)"
  - Not necessarily an easy function to compute, mind you
- Consider \( X \rightarrow h \), where \( h \) is the hash of attributes in \( X \)
- Notational conventions
  - "a", "b", "c" – specific attributes
  - "A", "B", "C" – sets of (unnamed) attributes
  - \( abc \rightarrow def \) – same as \( (a,b,c) \rightarrow (d,e,f) \)

Most common to see singletons (\( X \rightarrow y \) or \( abc \rightarrow d \))

FD: relaxes the concept of a “key”

- Functional dependency: \( X \rightarrow Y \)
- Superkey: \( X \rightarrow R \)
- A superkey must include all remaining attributes of the relation on the RHS
- An FD can involve just a subset of them
- Example:
  - Houses(street, city, value, owner, tax)
  - street,city \( \rightarrow \) value,owner,tax (both FD and key)
  - city,value \( \rightarrow \) tax (FD only)

Rules and principles about FDs

- Rules
  - The splitting/combining rule
  - Trivial FDs
  - The transitive rule
- Algorithms related to FDs
  - the closure of a set of attributes of a relation
  - a minimal basis of a relation
The Splitting/Combining rule of FDs

- Attributes on right independent of each other
  - Consider \(a, b, c \rightarrow d, e, f\)
  - “Attributes a, b, and c functionally determine d, e, and f”
  => No mention of d relating to e or f directly

- Splitting rule (Useful to split up right side of FD)
  - \(abc \rightarrow def\) becomes \(abc \rightarrow d, abc \rightarrow e\) and \(abc \rightarrow f\)

- No safe way to split left side
  - \(abc \rightarrow def\) is NOT the same as \(ab \rightarrow def\) and \(c \rightarrow def\)

- Combining rule (Useful to combine right sides):
  - if \(abc \rightarrow d, abc \rightarrow e, abc \rightarrow f\) holds, then \(abc \rightarrow def\) holds

Splitting FDs – example

- Consider the relation and FD
  - EmailAddress(user, domain, firstName, lastName)
  - user, domain \rightarrow firstName, lastName

- The following hold
  - user, domain \rightarrow firstName
  - user, domain \rightarrow lastName

- The following do NOT hold!
  - user \rightarrow firstName, lastName
  - domain \rightarrow firstName, lastName

  \textit{Gotcha: “doesn’t hold” = “not all tuples” != “all tuples not”}

Trivial FDs

- Not all functional dependencies are useful
  - A \rightarrow A always holds
  - \(abc \rightarrow a\) also always holds (right side is subset of left side)

- FD with an attribute on both sides is “trivial”
  - Simplify by removing \(L \cap R\) from \(R\)
    - \(abc \rightarrow ad\) becomes \(abc \rightarrow d\)
  - Or, in singleton form, delete trivial FDs
    - \(abc \rightarrow a\) and \(abc \rightarrow d\) becomes just \(abc \rightarrow d\)

Transitive rule

- The transitive rule holds for FDs
  - Consider the FDs: \(a \rightarrow b\) and \(b \rightarrow c\); then \(a \rightarrow c\) holds
  - Consider the FDs: \(ad \rightarrow b\) and \(b \rightarrow cd\); then \(ad \rightarrow cd\) holds or just \(ad \rightarrow c\) (because of the trivial dependency rule)
Identifying functional dependencies

- **FDs are domain knowledge**
  - Intrinsic features of the data you’re dealing with
  - Something you know (or assume) about the data
- **Database engine cannot identify FDs for you**
  - Designer must specify them as part of schema
  - DBMS can only enforce FDs when told to
- **DBMS cannot safely “optimize” FDs either**
  - It has only a finite sample of the data
  - An FD constrains the entire domain

### Coincidence or FD?

<table>
<thead>
<tr>
<th>ID</th>
<th>Email</th>
<th>City</th>
<th>Country</th>
<th>Surname</th>
</tr>
</thead>
<tbody>
<tr>
<td>1983</td>
<td><a href="mailto:tom@gmail.com">tom@gmail.com</a></td>
<td>Toronto</td>
<td>Canada</td>
<td>Fairgrieve</td>
</tr>
<tr>
<td>8624</td>
<td><a href="mailto:mar@bell.com">mar@bell.com</a></td>
<td>London</td>
<td>Canada</td>
<td>Samways</td>
</tr>
<tr>
<td>9141</td>
<td><a href="mailto:scotty@gmail.com">scotty@gmail.com</a></td>
<td>Winnipeg</td>
<td>Canada</td>
<td>Samways</td>
</tr>
<tr>
<td>1204</td>
<td><a href="mailto:birds@gmail.com">birds@gmail.com</a></td>
<td>Aachen</td>
<td>Germany</td>
<td>Lakemeyer</td>
</tr>
</tbody>
</table>

- **What if we try to infer FDs from the data?**
  - ID -> email, city, country, surname
  - email -> city, country, surname
  - city -> country
  - surname -> country

*Domain knowledge required to validate FDs*

### Keys and FDs

- **Consider relation R with attributes A**
- **Superkey**
  - Any $S \subseteq A$ s.t. $S \rightarrow A$
  - Any subset of A which determines all remaining attributes in A
- **Candidate key (or key)**
  - $C \subseteq A$ s.t. $C \rightarrow A$ and $X \rightarrow A$ does not hold for any $X \subseteq C$
  - A superkey which contains no other superkeys
  - Remove any attribute and you no longer have a key
- **Primary key**
  - The candidate key we use to identify the relation
  - Always exists, only one allowed, doesn’t matter which C we use
- **Prime attribute**
  - $\exists$ candidate key $C$ s.t. $x \in C$ (attribute that participates in at least one key)

### Candidate keys vs. superkeys

- **Consider these relations**
  - Students(ID, surname, name, email, address, major)
  - Houses(street, city, value, owner, tax)
- **What are the candidate keys?**
  - Students: ID, what else?
  - Houses?:
- **What other superkeys exist?**
  - Students: ID,surname ID,name ID,name,surname ...
  - Houses?:
- **Prime attributes?**
  - Students: ?
  - Houses?:

Cyclic functional dependencies?

- Attributes on right side of one FD may appear on left side of another!
  - Simple example: assume relation (A, B) & FDs: A -> B, B -> A
  - What does this say about A and B?
- Example
  - studentID -> email  email -> studentID

Inferring functional dependencies

- Problem
  - Given FDs X₁ -> a₁, X₂ -> a₂, etc.
  - Does some FD Y -> B (not given) also hold?
- Consider the dependencies
  A -> B  B -> C
  Intuitively, A -> C also holds
  The given FDs entail (imply) it (transitivity rule)

How to prove it in the general case?

Geometric view of FDs

- Let D be the domain of tuples in R
  - Every possible tuple is a point in D
- FD X on R restricts tuples in R to a subset of D
  - Points in D which violate X cannot be in R
- Example: D(x,y,z)
  - x y -> z
  - z -> x,y
  - => x = y = abs(z)/2

Closure test for FDs

- Given attribute set A and FD set F
  - Denote $A_F^+$ as the closure of A relative to F
    $=> A_F^+ = \text{set of all FDs given or implied by } A$
- Computing the [transitive] closure of A
  - Start: $A_F^+ = A, F' = F$
  - While $\exists X \in F'$ s.t. LHS(X) $\subseteq A_F^+$:
    $A_F^+ = A_F^+ \cup \text{RHS}(X)$
    $F' = F' - X$
  - At end: $A -> B \forall B \in A_F^+$
Closure test – example

- Consider R(a,b,c,d,e,f) with FDs ab -> c, ac -> d, c -> e, ade -> f
- Find A⁺ if A = ab or find \(\{a,b\}⁺\)

\[
\begin{align*}
\{a,b\}⁺ &= \{a,b,c,d,e,f\} \\
ab \quad \text{or} \quad cdef \\
\end{align*}
\]

\(\{a,b\}⁺\) is a candidate key!

Example : Closure Test

\[
\begin{align*}
F: & \ AB \rightarrow C \\
& A \rightarrow D \\
& D \rightarrow E \\
& AC \rightarrow B
\end{align*}
\]

\[
\begin{array}{c|c}
X & X⁺ \\
\hline
A & \{A, D, E\} \\
AB & \{A, B, C, D, E\} \\
AC & \{A, C, B, D, E\} \\
B & \{B\} \\
D & \{D, E\}
\end{array}
\]

Is \(AB \rightarrow E\) entailed by \(F\)? Yes
Is \(D \rightarrow C\) entailed by \(F\)? No

Result: \(X⁺\) allows us to determine all FDs of the form \(X \rightarrow Y\) entailed by \(F\)

Discarding redundant FDs

- Minimal basis: opposite extreme from closure
- Given a set of FDs \(F\), want to minimize \(F'\) s.t.
  - \(F' \subseteq F\)
  - \(F'\) entails \(X \forall X \in F\)
- Properties of a minimal basis \(F'\)
  - 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 \(F\), the result is no longer a minimal basis

Constructing a minimal basis

- Straightforward but time-consuming
  1. Split all RHS into singletons
  2. \(\forall X \in F', \text{ test whether } J = (F' - X)⁺ \text{ is still equivalent to } F⁺\)
     \(\Rightarrow\) Might make \(F'\) too small
  3. \(\forall i \in \text{LHS}(X) \forall X \in F', \text{ let } \text{LHS}(X') = \text{LHS}(X) - i\)
     Test whether \((F' - X + X')⁺\) is still equivalent to \(F⁺\)
     \(\Rightarrow\) Might make \(F'\) too big
  4. Repeat (2) and (3) until neither makes progress
Minimal Basis: Example

• Relation R: R(A, B, C, D)
• Defined FDs:
  – F = {A->AC, B->ABC, D->ABC}

Find the minimal Basis M of F

Minimal Basis: Example (cont.)

1st Step

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 outcome => H = {A->C, B->A, D->B}

Minimal Basis: Example (cont.)

3rd Step
– H doesn’t change as all LHS in H are single attributes

4th Step
– H doesn’t change

Minimal Basis: M = H = {A->C, B->A, D->B}

Minimal Basis: Example 2

• Relation R: R(A, B, C)
• Defined FDs:
  – A->B, A->C, B->C, B->A, C->A, C->B
  – CA->B, AC->B, BC->A
  – A->BC
  – A->A

• Possible Minimal Bases:
  – {A->B, B->A, B->C, C->B} or
  – {A->B, B->C, C->A}
  – ...
Representing FDs as graphs

• Insight: treat an FD as a directed edge in a graph
  – Entire LHS becomes a “closed” node (or node cluster)
  – Each attribute of RHS becomes an “open” node
  – Draw edge from LHS to RHS
  – OK to merge open node(s) with a matching closed node
  => *Illegal to merge open nodes with each other directly*

• Terminology in terms of graphs
  – Superkey: set of nodes which reaches all sinks
  – Candidate key: any non-redundant set of sources which reaches all sinks (e.g. removing any source orphans 1+ sinks)
  => *Source node <=/= prime attribute*

Example: FD set as graph

• Example 1: a->bc  b->c  d->b

• Example 2: ab->c  c->d  ce -> a

FDs and redundancy

• Given relation R and FDs F
  – R often exhibits anomalies due to redundancy
  – F identifies many (not all) of the underlying problems

• Idea
  – Use F to identify “good” ways to split relations
  – Split R into 2+ smaller relations having less redundancy
  – Split up F into subsets which apply to the new relations
    (compute the projection of functional dependencies)
Schema decomposition

- Given relation $R$ and FDs $F$
  - Split $R$ into $R_i$ s.t. $\forall i R_i \subseteq R$ (no new attributes)
  - Split $F$ into $F_i$ s.t. $\forall i F_i$ entails $F_i$ (no new FDs)
  - $F_i$ involves only attributes in $R_i$
- Caveat: entirely possible to lose information
  - $F^*$ may entail $X$ which is not in $(U_i F_i)^*$
    => Decomposition lost some FDs
  - Possible to have $R \subseteq R_i$, $R_i$
    => Decomposition lost some relationships
- Goal: minimize anomalies without losing info

*We’ll revisit information loss later*

Gotcha: lossy join decomposition

- Consider a relation with one more tuple

<table>
<thead>
<tr>
<th>Student Name</th>
<th>Student Email</th>
<th>Course</th>
<th>Instructor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xiao</td>
<td>xiao@gmail</td>
<td>CSCC43</td>
<td>Johnson</td>
</tr>
<tr>
<td>Xiao</td>
<td>xiao@gmail</td>
<td>CSCD08</td>
<td>Bretscher</td>
</tr>
<tr>
<td>Jaspreet</td>
<td>jaspreet@utsc</td>
<td>SCSC43</td>
<td>Johnson</td>
</tr>
<tr>
<td>Mary</td>
<td>mary@utsc</td>
<td>CSCD08</td>
<td>Rosenburg</td>
</tr>
</tbody>
</table>

- **Students $\bowtie$ Taking $\bowtie$ Courses** has bogus tuples!
  - Mary is not taking Bretscher’s section of D08
  - Xiao is not in Rosenburg’s section of D08

*Why did this happen? How to prevent it?*

Splitting relations – example

- Consider the following relation:

<table>
<thead>
<tr>
<th>Student Name</th>
<th>Student Email</th>
<th>Course</th>
<th>Instructor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xiao</td>
<td>xiao@gmail</td>
<td>CSCC43</td>
<td>Johnson</td>
</tr>
<tr>
<td>Xiao</td>
<td>xiao@gmail</td>
<td>CSCD08</td>
<td>Bretscher</td>
</tr>
<tr>
<td>Jaspreet</td>
<td>jaspreet@utsc</td>
<td>SCSC43</td>
<td>Johnson</td>
</tr>
</tbody>
</table>

- One possible decomposition
  - Students(email, name)
  - Courses(name, instructor)
  - Taking(studentEmail, courseName)

Information loss with decomposition

- Decompose $R$ into $S$ and $T$
  - Consider FD $a \rightarrow b$, with $a$ only in $S$ and $b$ only in $T$
- FD loss
  - Attributes $a$ and $b$ no longer in same relation
    => Must join $T$ and $S$ to enforce $a \rightarrow b$ (expensive)
- Join loss
  - LHS and RHS no longer in same relation, no other connection
    - Neither $(S \cap T) \rightarrow S$ nor $(S \cap T) \rightarrow T$ in $F^*$
    => Joining $T$ and $S$ produces bogus tuples (irreparable)
- In our example:
  - $(\{email, course\} \cap \{course, instructor\}) = \{course\}$
  - course $\rightarrow$ instructor and course $\rightarrow$ email
FD loss as a graph

Joining recovers original relation because `studio -> studioAddr`

Projecting FDs

- Once we’ve split a relation we have to refactor our FDs to match
  - Each FDs 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)

FD projection algorithm

- Start with $F_1 = \emptyset$
- For each subset $X$ of $R_i$
  - Compute $X^+$
  - For each attribute $a$ in $X^+$
    - If $a$ is in $R_i$
      - add $X \to a$ to $F_i$
- Compute the minimal basis of $F_i$
- Projection is expensive
  - Suppose $R_j$ has $n$ attributes
  - How many subsets of $R_j$ are there?
Making projection more efficient

• Ignore trivial dependencies
  – No need to add \( X \rightarrow A \) if \( A \) is in \( X \) itself

• 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 FD’s

• \( ABC \) with FD’s \( A \rightarrow B \) and \( B \rightarrow C \)
  – \( A^+ = ABC \); yields \( A \rightarrow B, A \rightarrow C \)
    • We do not need to compute \( AB^+ \) or \( AC^+ \)
  – \( B^+ = BC \); yields \( B \rightarrow C \)
  – \( C^+ = C \); yields nothing.
  – \( BC^+ = BC \); yields nothing.

Example -- Continued

• Resulting FD’s: \( A \rightarrow B, A \rightarrow C, \) 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 \( \{B, C\} \)

Part III:
Normal forms
Motivation for normal forms

• Identify a “good” schema
  – For some definition of “good”
  – Avoid anomalies, redundancy, etc.

• Many normal forms
  – 1st
  – 2nd
  – 3rd
  – Boyce-Codd
  – ... and several more we won’t discuss...

BCNF ⊆ 3NF ⊆ 2NF ⊆ 1NF (focus on 3NF/BCNF)

1st normal form (1NF)

• No multi-valued attributes allowed
  – Imagine storing a list/set of things in an attribute
  => Not really even expressible in RA

• Counterexample
  – Course(name, instructor, [student,email]*)
  – Redundancy in non-list attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Instructor</th>
<th>Student Name</th>
<th>Student Email</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSCC43</td>
<td>Johnson</td>
<td>Xiao</td>
<td>xiao@gmail</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Jaspreet</td>
<td>jaspreet@utsc</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mary</td>
<td>mary@utsc</td>
</tr>
<tr>
<td>CSCD08</td>
<td>Rosenberg</td>
<td>Jaspreet</td>
<td>jaspreet@utsc</td>
</tr>
</tbody>
</table>

1NF in terms of graphs?

• We only need to graph the schema
  => Structure of tuples does not vary from tuple to tuple

• Consider again our example
  => Cannot capture the structure at schema level only

<table>
<thead>
<tr>
<th>Name</th>
<th>Instructor</th>
<th>Student Name</th>
<th>Student Email</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSCC43</td>
<td>Johnson</td>
<td>Xiao</td>
<td>xiao@gmail</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Jaspreet</td>
<td>jaspreet@utsc</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mary</td>
<td>mary@utsc</td>
</tr>
<tr>
<td>CSCD08</td>
<td>Rosenberg</td>
<td>Jaspreet</td>
<td>jaspreet@utsc</td>
</tr>
</tbody>
</table>

2nd normal form (2NF)

• Non-prime attributes depend on candidate keys
  – Consider non-prime (ie. not part of a key) attribute ‘a’
  – Then ∃FD X s.t. X -> a and X is a candidate key

• Counterexample
  – Movies(title, year, star, studio, studioAddress, salary)
  – FD: title, year -> studio; studio -> studioAddress; star->salary

<table>
<thead>
<tr>
<th>Title</th>
<th>Year</th>
<th>Star</th>
<th>Studio</th>
<th>StudioAddr</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Star Wars</td>
<td>1977</td>
<td>Hamill</td>
<td>Lucasfilm</td>
<td>1 Lucas Way</td>
<td>$100,000</td>
</tr>
<tr>
<td>Star Wars</td>
<td>1977</td>
<td>Ford</td>
<td>Lucasfilm</td>
<td>1 Lucas Way</td>
<td>$100,000</td>
</tr>
<tr>
<td>Star Wars</td>
<td>1977</td>
<td>Fisher</td>
<td>Lucasfilm</td>
<td>1 Lucas Way</td>
<td>$100,000</td>
</tr>
<tr>
<td>Patriot Games</td>
<td>1992</td>
<td>Ford</td>
<td>Paramount</td>
<td>Cloud 9</td>
<td>$2,000,000</td>
</tr>
<tr>
<td>Last Crusade</td>
<td>1989</td>
<td>Ford</td>
<td>Lucasfilm</td>
<td>1 Lucas Way</td>
<td>$1,000,000</td>
</tr>
</tbody>
</table>
2NF in terms of graphs

- Require a path from every source to every sink
  - No trivial edges allowed!
  - Disconnected components violate 2NF
  - Watch for node clusters which are subsets of candidate keys

3rd normal form (3NF)

- Non-prime attr. depend only on candidate keys
  - Consider FD X -> a
  - Either a ∈ X OR X is a superkey OR a is prime (part of a key)
  => No transitive dependencies allowed

Counterexample:
- studio -> studioAddr
  (studioAddr depends on studio which is not a candidate key)

<table>
<thead>
<tr>
<th>Title</th>
<th>Year</th>
<th>Studio</th>
<th>StudioAddr</th>
</tr>
</thead>
<tbody>
<tr>
<td>Star Wars</td>
<td>1977</td>
<td>Lucasfilm</td>
<td>1 Lucas Way</td>
</tr>
<tr>
<td>Patriot Games</td>
<td>1992</td>
<td>Paramount</td>
<td>Cloud 9</td>
</tr>
<tr>
<td>Last Crusade</td>
<td>1989</td>
<td>Lucasfilm</td>
<td>1 Lucas Way</td>
</tr>
</tbody>
</table>

3NF in terms of graphs

- 3NF violation: transitive dependency

3NF, dependencies, and join loss

- Theorem: always possible to convert a schema to join-lossless, dependency-preserving 3NF
- Caveat: always possible to create schemas in 3NF for which these properties do not hold

Join 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)
  => Cannot enforce Movies⇒StarSalary yields bogus tuples (irreparable)

Note: OK for decomposition to lose redundant FDs
Graphs and lossy decomposition

- Loss: an FD which spans two relations
  - Join loss if no transitive connection between the two nodes
    => No set of joins can reconstruct the connection
- Our 3NF example showed a lost dependency
  - title, year -> studioAddr
    => No join loss because title->year -> studioName -> studioAddr

Boyce-Codd normal form (BCNF)

- One additional restriction over 3NF
  - All non-trivial FD 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 non-prime
    - Violates BCNF: postalCode is not a superkey
    => Possible anomalies involving postalCode

Do we care? How often do postal codes change?

Another Example

```sql
emps(emp_id, emp_name, emp_phone, dept_name, emp_city, emp_straddr)
empadds (emp_city, emp_zip, emp_straddr)
```

FDs:
- emp_id -> emp_name emp_phone dept_dname
- emp_city emp_straddr -> emp_zip
- emp_zip -> emp_city

The FD emp_zip -> emp_city is preserved in the relation empadds but emp_zip is not a key. The schema is not in BCNF.

The attribute emp_city is prime (there is key emp_city emp_straddr). Hence the schema is in 3NF.

More Examples

```sql
<table>
<thead>
<tr>
<th>Manager</th>
<th>Project</th>
<th>Branch</th>
<th>Division</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brown</td>
<td>Mars</td>
<td>Chicago</td>
<td>1</td>
</tr>
<tr>
<td>Green</td>
<td>Jupiter</td>
<td>Birmingham</td>
<td>1</td>
</tr>
<tr>
<td>Green</td>
<td>Mars</td>
<td>Birmingham</td>
<td>1</td>
</tr>
<tr>
<td>Hoskins</td>
<td>Saturn</td>
<td>Birmingham</td>
<td>2</td>
</tr>
<tr>
<td>Hoskins</td>
<td>Venus</td>
<td>Birmingham</td>
<td>2</td>
</tr>
</tbody>
</table>
```

- Functional dependencies:
  - Manager -> Branch, Division -- each manager works at one branch and manages one division
  - Branch, Division -> Manager -- for each branch and division there is a single manager
  - Project, Branch -> Division, Manager -- for each branch, a project is allocated to a single division and has a sole manager responsible
A good decomposition

- Note: The first relation has a second key \{\text{Branch}, \text{Division}\}
- The decomposition is in 3NF but not in BCNF; moreover, it is lossless and dependencies are preserved
- This example demonstrates that BCNF is too strong a condition to impose on a relational schema

<table>
<thead>
<tr>
<th>Manager</th>
<th>Branch</th>
<th>Division</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brown</td>
<td>Chicago</td>
<td>1</td>
</tr>
<tr>
<td>Green</td>
<td>Birmingham</td>
<td>1</td>
</tr>
<tr>
<td>Hoskins</td>
<td>Birmingham</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Project</th>
<th>Branch</th>
<th>Division</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mars</td>
<td>Chicago</td>
<td>1</td>
</tr>
<tr>
<td>Jupiter</td>
<td>Birmingham</td>
<td>1</td>
</tr>
<tr>
<td>Mars</td>
<td>Birmingham</td>
<td>1</td>
</tr>
<tr>
<td>Saturn</td>
<td>Birmingham</td>
<td>2</td>
</tr>
<tr>
<td>Venus</td>
<td>Birmingham</td>
<td>2</td>
</tr>
</tbody>
</table>

Limits of decomposition

- Pick two...
  - Lossless join
  - Dependency preservation
  - Anomaly-free
- 3NF
  - Always allows join lossless and dependency preserving
  - May allow some anomalies
- BCNF
  - Always excludes anomalies
  - May give up one of join lossless or dependency preserving

*Use domain knowledge to choose 3NF vs. BCNF*