FUNCTIONAL DEPENDENCY THEORY II

CS121: Introduction to Relational Database Systems

Fall 2014 – Lecture 20

Last Time: Canonical Cover

- Last time, introduced concept of canonical cover
- A canonical cover F_c for F is a set of functional dependencies such that:
  - F logically implies all dependencies in F_c
  - F_c logically implies all dependencies in F
  - Can't infer any functional dependency in F_c from other dependencies in F_c
  - No functional dependency in F_c contains an extraneous attribute
  - Left side of all functional dependencies in F_c are unique
    - There are no two dependencies $\alpha_1 \rightarrow \beta_1$ and $\alpha_2 \rightarrow \beta_2$ in F_c such that $\alpha_1 = \alpha_2$

Extraneous Attributes

- Given a set F of functional dependencies
  - An attribute in a functional dependency is <u>extraneous</u> if it can be removed from F without affecting closure of F
- Formally: given F, and $\alpha \rightarrow \beta$
  - If $A \in \alpha$, and F logically implies $(F - \{\alpha \rightarrow \beta\}) \cup \{(\alpha - A) \rightarrow \beta\}$, then A is extraneous
  - If $A \in \beta$, and $(F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$ logically implies F, then A is extraneous
    - i.e. generate a new set of functional dependencies F' by replacing $\alpha \rightarrow \beta$ with $\alpha \rightarrow (\beta - A)$
    - See if F' logically implies F

Testing Extraneous Attributes

- Given relation schema *R*, and a set *F* of functional dependencies that hold on *R*
- Attribute *A* in $\alpha \rightarrow \beta$
- If $A \in \alpha$ (i.e. *A* is on left side of the dependency), then let $\gamma = \alpha - \{A\}$
  - See if $\gamma \rightarrow \beta$ can be inferred from *F*
  - Compute γ^+ under *F*
  - If $\beta \subseteq \gamma^+$ then *A* is extraneous in α

Testing Extraneous Attributes (2)

- Given relation schema *R*, and a set *F* of functional dependencies that hold on *R*
- Attribute *A* in $\alpha \rightarrow \beta$
- If $A \in \beta$ (on right side of the dependency), then try the altered set F'
  - $F' = (F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$
  - See if $\alpha \rightarrow A$ can be inferred from F'
  - Compute α^+ under F'
  - If α^+ includes *A* then *A* is extraneous in β

Computing Canonical Cover

- A simple way to compute the canonical cover of F

$F_c = F$
repeat
apply union rule to replace dependencies in F_c of form
$\alpha_1 \rightarrow \beta_1$ and $\alpha_1 \rightarrow \beta_2$ with $\alpha_1 \rightarrow \beta_1\beta_2$
find a functional dependency $\alpha \rightarrow \beta$ in F_c with an
extraneous attribute
/* Use F_c for the extraneous attribute test, not F !!! */
if an extraneous attribute is found, delete it from $\alpha \rightarrow \beta$
until F_c stops changing

Canonical Cover Example

- Functional dependencies F on schema (A, B, C)
  - $F = \{ A \rightarrow BC, B \rightarrow C, A \rightarrow B, AB \rightarrow C \}$
  - Find F_c
- Apply union rule to $A \rightarrow BC$ and $A \rightarrow B$
  - Left with: $\{ A \rightarrow BC, B \rightarrow C, AB \rightarrow C \}$
- A is extraneous in $AB \rightarrow C$
  - $B \rightarrow C$ is logically implied by F (obvious)
  - Left with: $\{ A \rightarrow BC, B \rightarrow C \}$
- C is extraneous in $A \rightarrow BC$
  - Logically implied by $A \rightarrow B$, $B \rightarrow C$
- $F_c = \{ A \rightarrow B, B \rightarrow C \}$

Canonical Covers

- A set of functional dependencies can have multiple canonical covers
- Example:
  - $F = \{ A \rightarrow BC, B \rightarrow AC, C \rightarrow AB \}$
  - Has several canonical covers:
    - $F_c = \{ A \rightarrow B, B \rightarrow C, C \rightarrow A \}$
    - $F_c = \{ A \rightarrow B, B \rightarrow AC, C \rightarrow B \}$
    - $F_c = \{ A \rightarrow C, C \rightarrow B, B \rightarrow A \}$
    - $F_c = \{ A \rightarrow C, B \rightarrow C, C \rightarrow AB \}$
    - $F_c = \{ A \rightarrow BC, B \rightarrow A, C \rightarrow A \}$

Another Example

- Functional dependencies F on schema (A, B, C, D)
  - $F = \{ A \rightarrow B, BC \rightarrow D, AC \rightarrow D \}$
  - Find F_c
- In this case, it may look like $F_c = F$…
- However, can infer $AC \rightarrow D$ from $A \rightarrow B$, $BC \rightarrow D$ (pseudotransitivity), so $AC \rightarrow D$ is extraneous in F
  - Therefore, $F_c = \{ A \rightarrow B, BC \rightarrow D \}$
- Alternately, can argue that D is extraneous in $AC \rightarrow D$
  - With $F' = \{ A \rightarrow B, BC \rightarrow D \}$, we see that $\{AC\}^+ = ACD$, so D is extraneous in $AC \rightarrow D$
  - (If you eliminate the entire RHS of a functional dependency, it goes away)

Lossy Decompositions

- Some schema decompositions lose information
- Example:

  employee(*<u>emp_id</u>*, *emp_name*, *phone*, *title*, *salary*, *start_date*)

  - Decomposed into:

    emp_ids(*<u>emp_id</u>*, *emp_name*)

    emp_details(*emp_name*, *phone*, *title*, *salary*, *start_date*)
- Problem:
  - *emp_name* doesn't uniquely identify employees
  - This is a lossy decomposition

Lossless Decompositions

- Given:
  - Relation schema *R*, relation *r*(*R*)
  - Set of functional dependencies *F*
- Let R_1 and R_2 be a decomposition of *R*
  - $R_1 \cup R_2 = R$
- The decomposition is lossless if, for <u>all</u> legal instances of *r* :

$$\Pi_{R_1}(r) \bowtie \Pi_{R_2}(r) = r$$

- A simple definition…

Lossless Decompositions (2)

- Can define with functional dependencies:
  - R_1 and R_2 form a lossless decomposition of R if at least one of these dependencies is in F^+ :

    $R_1 \cap R_2 \rightarrow R_1$

    $R_1 \cap R_2 \rightarrow R_2$
- $R_1 \cap R_2$ forms a superkey of R_1 and/or R_2
  - Test for superkeys using attribute-set closure

Decomposition Examples (1)

- The *employee* example:

  employee(*emp_id*, *emp_name*, *phone*, *title*, *salary*, *start_date*)

- Decomposed into:

  emp_ids(*emp_id*, *emp_name*)

  emp_details(*emp_name*, *phone*, *title*, *salary*, *start_date*)

- *emp_name* is not a superkey of *emp_ids* or *emp_details*, so the decomposition is lossy

Decomposition Examples (2)

- The *bor_loan* example:

  bor_loan(*<u>cust_id</u>*, *<u>loan_id</u>*, *amount*)

- Decomposed into:

  borrower(*cust_id*, *loan_id*)

  loan(*<u>loan_id</u>*, *amount*)   (*loan_id* $\rightarrow$ *loan_id*, *amount*)

- *loan_id* is a superkey of *loan*, so the decomposition is lossless

BCNF Decompositions

- If R is a schema not in BCNF:
  - There is at least one nontrivial functional dependency $\alpha \rightarrow \beta$ such that α is not a superkey for R
  - For simplicity, also require that $\alpha \cap \beta = \varnothing$
    - (if $\alpha \cap \beta \neq \varnothing$ then $(\alpha \cap \beta)$ is extraneous in β)
- Replace R with two schemas:

  $R_1 = (\alpha \cup \beta)$

  $R_2 = (R - \beta)$
    - (was $R - (\beta - \alpha)$, but $\beta - \alpha = \beta$, since $\alpha \cap \beta = \varnothing$)
- BCNF decomposition is lossless
  - $R_1 \cap R_2 = \alpha$
  - α is a superkey of R_1
  - α also appears in R_2

Dependency Preservation

- Some schema decompositions are not dependency-preserving
  - Functional dependencies that span multiple relation schemas are hard to enforce
  - e.g. BCNF may require decomposition of a schema for one dependency, and make it hard to enforce another dependency
- Can test for dependency preservation using functional dependency theory

Dependency Preservation (2)

- Given:
  - A set F of functional dependencies on a schema R
  - R_1, R_2, …, R_n are a decomposition of R
- The restriction of F to R_i is the set F_i of functional dependencies in F^+ that only has attributes in R_i
  - Each F_i contains functional dependencies that can be checked efficiently, using only R_i
- Find *all* functional dependencies that can be checked efficiently
  - $F' = F_1 \cup F_2 \cup \ldots \cup F_n$
  - If $F'^+ = F^+$ then the decomposition is dependency-preserving

Third Normal Form Schemas

- Can generate a 3NF schema from a set of functional dependencies *F*
- Called the <u>3NF synthesis algorithm</u>
  - Instead of decomposing an initial schema, generates schemas from a set of dependencies
- Given a set *F* of functional dependencies
  - Uses the canonical cover F_c
  - Ensures that resulting schemas are dependency-preserving

3NF Synthesis Algorithm

- Inputs: set of functional dependences F, on a schema R

let F_c be a canonical cover for F ;
i := 0;
for each functional dependency $\alpha \rightarrow \beta$ in F_c **do**
    if none of the schemas R_j, j = 1, 2, …, i contains $(\alpha \cup \beta)$ **then**
        $i := i + 1$;
        $R_i := (\alpha \cup \beta)$
    end if
done
if no schema R_j, j = 1, 2, …, i contains a candidate key for R **then**
    $i := i + 1$;
    R_i := any candidate key for R
end if
return $(R_1, R_2, \ldots, R_i)$

BCNF vs. 3NF

- Boyce-Codd Normal Form:
  - Eliminates more redundant information than 3NF
  - Some functional dependencies become expensive to enforce
    - The conditions to enforce involve multiple relations
  - Overall, a very desirable normal form!
- Third Normal Form:
  - All [more] dependencies are [probably] easy to enforce…
  - Allows more redundant information, which must be kept synchronized by the database application!
  - Personal banker example:

    works_in(<u>*emp_id*</u>, *branch_name*)

    cust_banker_branch(<u>*cust_id*</u>, <u>*branch_name*</u>, *emp_id*, *type*)

    - Branch names must be kept synchronized between these relations!

BCNF and 3NF vs. SQL

- SQL constraints:
  - Only <u>key</u> constraints are fast and easy to enforce!
  - Only easy to enforce functional dependencies $\alpha \rightarrow \beta$ if α is a key on some table!
  - Other functional dependencies (even "easy" ones in 3NF) may require more expensive constraints, e.g. **`CHECK`**
- For SQL databases with materialized views:
  - Can decompose a schema into BCNF
  - For dependencies $\alpha \rightarrow \beta$ not preserved in decomposition, create materialized view joining all relations in dependency
  - Enforce **unique**(α) constraint on materialized view
- Impacts both space and performance, but it works…

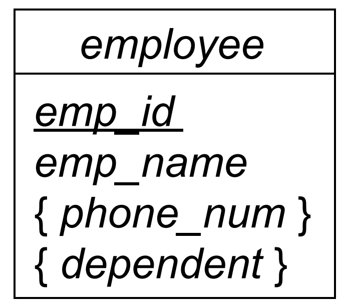
Multivalued Attributes

- E-R schemas can have multivalued attributes
- 1NF requires only atomic attributes
  - Not a problem; translating to relational model leaves everything atomic
- Employee example:

  employee(*emp_id*, *emp_name*)

  emp_deps(*emp_id*, *dependent*)

  emp_nums(*emp_id*, *phone_num*)



- What are the requirements on these schemas for what tuples must appear?

Multivalued Attributes (2)

- Example data:

emp_id	*emp_name*
125623	Rick

employee

emp_id	*dependent*
125623	Jeff
125623	Alice

emp_deps

emp_id	*phone_num*
125623	555-8888
125623	555-2222

emp_nums

  - Every distinct value of multivalued attribute requires a separate tuple, including associated value of *emp_id*
- A consequence of 1NF, in fact!
  - If attributes could be nonatomic, could just store list of values in the appropriate column!
  - 1NF <u>requires</u> extra tuples to represent multivalues

Independent Multivalued Attributes

- Question is trickier when a schema stores several *independent* multivalued attributes
- Proposed combined schema:

  employee(*emp_id*, *emp_name*)

  emp_info(*emp_id*, *dependent*, *phone_num*)
- What tuples must appear in *emp_info* ?
  - *emp_info* is a relation
  - If an employee has M dependents and N phone numbers, *emp_info* must contain M × N tuples
    - Exactly what we get if we natural-join *emp_deps* and *emp_nums*
  - Every combination of the employee's dependents and their phone numbers

Independent Multivalued Attributes

- Example data:

emp_id	*emp_name*
125623	Rick

employee

emp_id	*dependent*	*phone_num*
125623	Jeff	555-8888
125623	Jeff	555-2222
125623	Alice	555-8888
125623	Alice	555-2222

emp_info

- Clearly has unnecessary redundancy
- Can't formulate functional dependencies to represent multivalued attributes
- Can't use BCNF or 3NF decompositions to eliminate redundancy in these cases

Multivalued Attributes Example

- Two employees: Rick and Bob
  - Both share a phone number at work
  - Both have two kids
  - Both have a kid named Alice
- Can't use functional dependencies to reason about this situation!
  - *emp_id* → *phone_num* doesn't hold since an employee can have several phone numbers
  - *phone_num* → *emp_id* doesn't hold either, since several employees can have the same phone number
  - Same with *emp_id* and *dependent*…

emp_id	*emp_name*
125623	Rick
127341	Bob

employee

emp_id	*phone_num*
125623	555-8888
125623	555-2222
127341	555-2222

emp_nums

emp_id	*dependent*
125623	Jeff
125623	Alice
127341	Alice
127341	Clara

emp_deps

Dependencies

- Functional dependencies rule out what tuples can appear in a relation
  - If $A \rightarrow B$ holds, then tuples cannot have same value for A but different values for B
  - Also called equality-generating dependencies
- Multivalued dependencies specify what tuples must be present
  - To represent a multivalued attribute's values properly, a certain set of tuples *must* be present
  - Also called tuple-generating dependencies

Multivalued Dependencies

- Given a relation schema R
  - Attribute-sets $\alpha \in R$, $\beta \in R$
  - $\alpha \twoheadrightarrow \beta$ is a multivalued dependency
  - "α multidetermines β"
- A multivalued dependency $\alpha \twoheadrightarrow \beta$ holds on R if, in any legal relation $r(R)$:

  For all pairs of tuples t_1 and t_2 in r such that $t_1[\alpha] = t_2[\alpha]$,
  There also exists tuples t_3 and t_4 in r such that:
  - $t_1[\alpha] = t_2[\alpha] = t_3[\alpha] = t_4[\alpha]$
  - $t_1[\beta] = t_3[\beta]$ and $t_2[\beta] = t_4[\beta]$
  - $t_1[R - \beta] = t_4[R - \beta]$ and $t_2[R - \beta] = t_3[R - \beta]$

Multivalued Dependencies (2)

- Multivalued dependency $\alpha \twoheadrightarrow \beta$ holds on *R* if, in any legal relation *r*(*R*):

  For all pairs of tuples t_1 and t_2 in *r* such that $t_1[\alpha] = t_2[\alpha]$,

  There also exists tuples t_3 and t_4 in *r* such that:

  - $t_1[\alpha] = t_2[\alpha] = t_3[\alpha] = t_4[\alpha]$
  - $t_1[\beta] = t_3[\beta]$ and $t_2[\beta] = t_4[\beta]$
  - $t_1[R - \beta] = t_4[R - \beta]$ and $t_2[R - \beta] = t_3[R - \beta]$
- Pictorially:

	α	β	$R - (\alpha \cup \beta)$
t_1	$a_1 \ldots a_i$	$a_{i+1} \ldots a_j$	$a_{j+1} \ldots a_n$
t_2	$a_1 \ldots a_i$	$b_{i+1} \ldots b_j$	$b_{j+1} \ldots b_n$
t_3	$a_1 \ldots a_i$	$a_{i+1} \ldots a_j$	$b_{j+1} \ldots b_n$
t_4	$a_1 \ldots a_i$	$b_{i+1} \ldots b_j$	$a_{j+1} \ldots a_n$

Multivalued Dependencies (3)

- Multivalued dependency:

	α	β	$R - (\alpha \cup \beta)$
t_1	$a_1 \ldots a_i$	$a_{i+1} \ldots a_j$	$a_{j+1} \ldots a_n$
t_2	$a_1 \ldots a_i$	$b_{i+1} \ldots b_j$	$b_{j+1} \ldots b_n$
t_3	$a_1 \ldots a_i$	$a_{i+1} \ldots a_j$	$b_{j+1} \ldots b_n$
t_4	$a_1 \ldots a_i$	$b_{i+1} \ldots b_j$	$a_{j+1} \ldots a_n$

- If $\alpha \twoheadrightarrow \beta$ then $R - (\alpha \cup \beta)$ is independent of this fact
  - Every distinct value of β must be associated once with every distinct value of $R - (\alpha \cup \beta)$
- Let $\gamma = R - (\alpha \cup \beta)$
  - If $\alpha \twoheadrightarrow \beta$ then also $\alpha \twoheadrightarrow \gamma$
  - $\alpha \twoheadrightarrow \beta$ implies $\alpha \twoheadrightarrow \gamma$
  - Sometimes written $\alpha \twoheadrightarrow \beta \mid \gamma$

Trivial Multivalued Dependencies

- $\alpha \twoheadrightarrow \beta$ is a trivial multivalued dependency on R if <u>all</u> relations $r(R)$ satisfy the dependency
- Specifically, $\alpha \twoheadrightarrow \beta$ is trivial if $\beta \subseteq \alpha$, or if $\alpha \cup \beta = R$
- Employee examples:
  - For schema *emp_deps*(*emp_id*, *dependent*), *emp_id* $\twoheadrightarrow$ *dependent* is trivial
  - For *emp_info*(*emp_id*, *dependent*, *phone_num*), *emp_id* $\twoheadrightarrow$ *dependent* is <u>not</u> trivial

Inference Rules

- Can reason about multivalued dependencies, just like functional dependencies
  - There is a set of complete, sound inference rules for MVDs
- Example inference rules:
  - Complementation rule:
    - If $\alpha \twoheadrightarrow \beta$ holds on R, then $\alpha \twoheadrightarrow R - (\alpha \cup \beta)$ holds
  - Multivalued augmentation rule:
    - If $\alpha \twoheadrightarrow \beta$ holds, and $\gamma \subseteq R$, and $\delta \subseteq \gamma$, then $\gamma\alpha \twoheadrightarrow \delta\beta$ holds
  - Multivalued transitivity rule:
    - If $\alpha \twoheadrightarrow \beta$ and $\beta \twoheadrightarrow \gamma$ holds, then $\alpha \twoheadrightarrow \gamma - \beta$ holds
  - Coalescence rule:
    - If $\alpha \twoheadrightarrow \beta$ holds, and $\gamma \subseteq \beta$, and there is a δ such that $\delta \subseteq R$, and $\delta \cap \beta = \varnothing$, and $\delta \rightarrow \gamma$, then $\alpha \rightarrow \gamma$ holds

Functional Dependencies

- Functional dependencies are also multivalued dependencies
- Replication rule:
  - If $\alpha \rightarrow \beta$, then $\alpha \twoheadrightarrow \beta$ too
  - Note there is an <u>additional</u> constraint from $\alpha \rightarrow \beta$: each value of α has *at most* one associated value for β
- Usually, functional dependencies are not stated as multivalued dependencies
  - The extra caveat is *important*, but not obvious in notation
  - Also, functional dependencies are easier to reason about!

Closures and Restrictions

- For a set D of functional and multivalued dependencies, can compute closure D^+
  - Use inference rules for both functional and multivalued dependencies to compute closure
- Sometimes need the restriction of D^+ to a relation schema R, too
- The restriction of D to a schema R_i includes:
  - All functional dependencies in D^+ that include only attributes in R_i
  - All multivalued dependencies of the form $\alpha \twoheadrightarrow \beta \cap R_i$, where $\alpha \subseteq R_i$, and $\alpha \twoheadrightarrow \beta$ is in D^+

Fourth Normal Form

- Given:
  - Relation schema *R*
  - Set of functional and multivalued dependencies *D*
- *R* is in 4NF with respect to *D* if:
  - For all multivalued dependencies $\alpha \twoheadrightarrow \beta$ in D^+, where $\alpha \subseteq R$ and $\beta \subseteq R$, at least one of the following holds:
    - $\alpha \twoheadrightarrow \beta$ is a trivial multivalued dependency
    - α is a superkey for *R*
  - Note: If $\alpha \rightarrow \beta$ then $\alpha \twoheadrightarrow \beta$
- A database design is in 4NF if all schemas in the design are in 4NF

4NF and BCNF

- Main difference between 4NF and BCNF is use of multivalued dependencies instead of functional dependencies
- Every schema in 4NF is also in BCNF
  - If a schema is not in BCNF then there is a nontrivial functional dependency $\alpha \rightarrow \beta$ such that α is not a superkey for R
  - If $\alpha \rightarrow \beta$ then $\alpha \twoheadrightarrow \beta$

4NF Decompositions

- Decomposition rule very similar to BCNF
- If schema *R* is not in 4NF with respect to a set of multivalued dependencies *D* :
  - There is some nontrivial dependency $\alpha \twoheadrightarrow \beta$ in D^+ where $\alpha \subseteq R$ and $\beta \subseteq R$, and α is not a superkey of *R*
    - Also constrain that $\alpha \cap \beta = \varnothing$
  - Replace *R* with two new schemas:
    - $R_1 = (\alpha \cup \beta)$
    - $R_2 = (R - \beta)$

Employee Information Example

- Combined schema:

  employee(<u>*emp_id*</u>, *emp_name*)

  emp_info(*emp_id*, *dependent*, *phone_num*)

  - Also have these dependencies:
    - *emp_id* → *emp_name*
    - *emp_id* →→ *dependent*
    - *emp_id* →→ *phone_num*
- *emp_info* is not in 4NF
- Following the rules for 4NF decomposition produces:

  (*emp_id*, *dependent*)

  (*emp_id*, *phone_num*)

  - Note: Each relation's candidate key is the entire relation. The multivalued dependencies are trivial.

Lossless Decompositions

- Can also define lossless decomposition with multivalued dependencies
  - R_1 and R_2 form a lossless decomposition of R if at least one of these dependencies is in D^+ :

$$R_1 \cap R_2 \twoheadrightarrow R_1$$

$$R_1 \cap R_2 \twoheadrightarrow R_2$$

Beyond Fourth Normal Form?

- Additional normal forms with various constraints
- Example: join dependencies
- Given R, and a decomposition R_1 and R_2 where $R_1 \cup R_2 = R$:
  - The decomposition is lossless if, for all legal instances of $r(R)$, $\Pi_{R_1}(r) \bowtie \Pi_{R_2}(r) = r$
- Can state this as a join dependency: $*(R_1, R_2)$
  - This is actually identical to a multivalued dependency!
  - $*(R_1, R_2)$ is equivalent to $R_1 \cap R_2 \twoheadrightarrow R_1 \mid R_2$

Join Dependencies and 5NF

- Join dependencies (JD) are a generalization of multivalued dependencies (MVD)
  - Can specify JDs involving N relation schemas, $N \geq 2$
  - JDs are equivalent to MVDs when $N = 2$
  - Can easily construct JDs where $N > 2$, with no equivalent set of MVDs
- Project-Join Normal Form (a.k.a. PJNF or 5NF):
  - A relation schema R is in PJNF with respect to a set of join dependencies D if, for all JDs in D^+ of the form $*(R_1, R_2, \ldots, R_n)$ where $R_1 \cup R_2 \cup \ldots \cup R_n = R$, at least one of the following holds:
    - $*(R_1, R_2, \ldots, R_n)$ is a trivial join dependency
    - Every R_i is a superkey for R

Join Dependencies and 5NF (2)

- If a schema is in Project-Join Normal Form then it is also in 4NF (and thus, in BCNF)
  - Every multivalued dependency is also a join dependency
  - (Every functional dependency is also a multivalued dependency)
- One small problem:
  - There isn't a complete, sound set of inference rules for join dependencies!
  - Can't reason about our set of join dependencies *D*…
  - This limits PJNF's real-world usefulness

Domain-Key Normal Form

- Domain-key normal form (DKNF) is an even more general normal form, based on:
  - **Domain constraints:** what values may be assigned to attribute *A*
    - Usually inexpensive to test, even with **`CHECK`** constraints
  - **Key constraints:** all attribute-sets *K* that are a superkey for a schema *R* (i.e. $K \rightarrow R$)
    - Almost always inexpensive to test
  - **General constraints:** other predicates on valid relations in a schema
    - Could be <u>very</u> expensive to test!
- A schema *R* is in DKNF if the domain constraints and key constraints logically imply the general constraints
  - An "ideal" normal form difficult to achieve in practice…