BKM_DATS: Databázové systémy
8. Relational DB Design

Vlastislav Dohnal

Relational Database Design

- Features of Good Relational Design
- Atomic Domains and First Normal Form

- Decomposition Using Functional Dependencies
- Functional Dependency Theory

- Algorithms for Functional Dependencies

Combine Schemas?

- Suppose we combine *instructor(ID, name, salary, dept_name)* and *department(dept_name, building, budget)* into *inst_dept*
  - No connection to a relationship set *inst_dept* !
- Result is possible repetition of information

ID	*name*	*salary*	*dept_name*	*building*	*budget*
22222	Einstein	95000	Physics	Watson	70000
12121	Wu	90000	Finance	Painter	120000
32343	El Said	60000	History	Painter	50000
45565	Katz	75000	Comp. Sci.	Taylor	100000
98345	Kim	80000	Elec. Eng.	Taylor	85000
76766	Crick	72000	Biology	Watson	90000
10101	Srinivasan	65000	Comp. Sci.	Taylor	100000
58583	Califieri	62000	History	Painter	50000
83821	Brandt	92000	Comp. Sci.	Taylor	100000
15151	Mozart	40000	Music	Packard	80000
33456	Gold	87000	Physics	Watson	70000
76543	Singh	80000	Finance	Painter	120000

What About Smaller Schemas?

- Suppose we had started with
  inst_dept (ID, name, salary, dept_name, building, budget)
  - How would we know to split up (**decompose**) it into *instructor* and *department*?
- Write a rule “if there were a schema (*dept_name, building, budget*), then *dept_name* would be a candidate key”
  - Denote as a **functional dependency**:

$$dept\_name \rightarrow building, budget$$

- In *inst_dept*, because *dept_name* is not a candidate key, the building and budget of a department may have to be repeated.
  - This indicates the need to decompose *inst_dept*

What About Smaller Schemas? (cont.)

- *inst_dept (ID, name, salary, dept_name, building, budget)*
- Not all decompositions are good.
  - Suppose we decompose *employee(ID, name, street, city, salary)* into
    - *instructor(ID, name, salary)* and *department(dept_name, building, budget)*

ID	*name*	*salary*
22222	Einstein	95000
12121	Wu	90000
32343	El Said	60000
45565	Katz	75000
98345	Kim	80000
76766	Crick	72000
10101	Srinivasan	65000
58583	Califieri	62000
83821	Brandt	92000
15151	Mozart	40000
33456	Gold	87000
76543	Singh	80000

dept_name	*building*	*budget*
Physics	Watson	70000
Finance	Painter	120000
History	Painter	50000
Comp. Sci.	Taylor	100000
Elec. Eng.	Taylor	85000
Biology	Watson	90000
Comp. Sci.	Taylor	100000
History	Painter	50000
Comp. Sci.	Taylor	100000
Music	Packard	80000
Physics	Watson	70000
Finance	Painter	120000

  - Do we lose information?
    - We cannot reconstruct the original *employee* relation.
    - This is a **lossy decomposition**.

A Lossy Decomposition

ID	*name*	*street*	*city*	*salary*
⋮				
57766	Kim	Main	Perryridge	75000
98776	Kim	North	Hampton	67000
⋮				

employee

ID	*name*
⋮	
57766	Kim
98776	Kim
⋮	

name	*street*	*city*	*salary*
⋮			
Kim	Main	Perryridge	75000
Kim	North	Hampton	67000
⋮			

natural join

ID	*name*	*street*	*city*	*salary*
⋮				
57766	Kim	Main	Perryridge	75000
57766	Kim	North	Hampton	67000
98776	Kim	Main	Perryridge	75000
98776	Kim	North	Hampton	67000
⋮				

Example of Lossless Decomposition

- **Lossless decomposition**
- Decomposition of

$R = (A, B, C)$ into $R_1 = (A, B)$ $R_2 = (B, C)$

A	*B*	*C*
α	1	A
β	2	B

r

A	*B*
α	*1*
β	*2*

$\Pi_{A,B}(r)$

B	*C*
1	A
2	B

$\Pi_{B,C}(r)$

$$r =^{?} \Pi_{A,B}(r) \bowtie \Pi_{B,C}(r)$$

A	*B*	*C*
α	1	A
β	2	B

Goal: Devise a Theory for the Following

- Decide whether a particular relation *R* is in a “good” form.
- In the case that a relation *R* is not in “good” form, decompose it into a set of relations $\{R_1, R_2, ..., R_n\}$ such that
  - each relation is in good form
  - the decomposition is a lossless decomposition

- Our theory is based on:
  - functional dependencies

Functional Dependencies

- Constraints on the set of legal relations.
- Require that the value for a particular set of attributes determines the value for another set of attributes uniquely.
  - E.g., employee_id determines employee name and address.
- A functional dependency is a generalization of the notion of a *key.*

Functional Dependencies (Cont.)

- Let *R* be a relation schema $\alpha \subseteq R$ *and* $\beta \subseteq R$ are non-empty
- The **functional dependency**

$$\alpha \rightarrow \beta$$

holds on *R* if and only if for any legal relation *r*(R), whenever any two tuples t_1 and t_2 of *r* agree on the attributes α, they also agree on the attributes β. That is,

$$t_1[\alpha] = t_2[\alpha] \Rightarrow t_1[\beta] = t_2[\beta]$$

- Read $\alpha \rightarrow \beta$ as *"β depends on α"*
- Example:
  - Consider *r*(*A,B*) with the following instance of *r.*

A	B
1	4
1	5
3	7

  - On this instance, $A \rightarrow B$ does **NOT** hold, but $B \rightarrow A$ does hold.

Use of Functional Dependencies

- We use functional dependencies to:
  - test relations to see if they are legal under a given set of functional dependencies.
    - If a relation *r* is legal under a set *F* of functional dependencies, we say that *r* **satisfies** *F.*
  - specify constraints on the set of legal relations
    - We say that *F* **holds on** *R* if all legal relations on *R* satisfy the set of functional dependencies *F.*
- Note
  - A specific instance of a relation schema may satisfy a functional dependency even if the functional dependency does not hold on all legal instances.
  - For example, a specific instance of *instructor(ID, name, salary)* may, by chance, satisfy

$$name \rightarrow ID.$$

Use of Functional Dependencies (Cont.)

- K is a superkey for a relation schema R if and only if $K \rightarrow R$
- K is a candidate key for R if and only if
  - $K \rightarrow R$, and
  - for no $\alpha \subset K$, $\alpha \rightarrow R$
- *Meaning: there is only one value for each value of K.*
- Functional dependencies allow us to express constraints that cannot be expressed using superkeys.
  - Consider the schema:
    inst_dept (<u>*ID*</u>, *name, salary, dept_name, building, budget*)
  - We expect these functional dependencies to hold:

    $dept\_name \rightarrow building$
    $ID \rightarrow building$
    $ID \rightarrow dept\_name$

    There is only one building for each department.

  but would not expect the following to hold:

    $dept\_name \rightarrow salary$

Functional Dependencies (Cont.)

- A functional dependency is **trivial** if it is satisfied by all instances of a relation
  - Example*:*
    - *ID, name* $\rightarrow$ *ID*
    - *name* $\rightarrow$ *name*
- In general, $\alpha \rightarrow \beta$ is trivial if $\beta \subseteq \alpha$

Closure of a Set of Functional Dependencies

- Given a set *F* of functional dependencies, there are certain other functional dependencies that are logically implied by *F*.
  - Example
    - If $A \rightarrow B$ and $B \rightarrow C$, then we can infer that $A \rightarrow C$
- The set of **all** functional dependencies logically implied by *F* is the **closure** of *F*.
  - We denote the *closure* of *F* by $\boldsymbol{F^+}$.
  - F^+ is a superset of *F*.

Closure of a Set of Functional Dependencies

- We can find F^{+}, the closure of F, by repeatedly applying **Armstrong's Axioms:**
  - if $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$ **(reflexivity)**
  - if $\alpha \rightarrow \beta$, then $\gamma\alpha \rightarrow \gamma\beta$ **(augmentation)**
  - if $\alpha \rightarrow \beta$, and $\beta \rightarrow \gamma$, then $\alpha \rightarrow \gamma$ **(transitivity)**
- These rules are
  - **sound** (generate only functional dependencies that actually hold), and
  - **complete** (generate all functional dependencies that hold).

Example

- $R = (A, B, C, G, H, I)$
  $F = \{ A \rightarrow B$
  $A \rightarrow C$
  $CG \rightarrow H$
  $CG \rightarrow I$
  $B \rightarrow H\}$
- some members of F^+
  - $A \rightarrow H$
    - by transitivity from $A \rightarrow B$ *and* $B \rightarrow H$
  - $AG \rightarrow I$
    - by augmenting $A \rightarrow C$ with G, to get $AG \rightarrow CG$
      and then transitivity with $CG \rightarrow I$
  - $CG \rightarrow HI$
    - by augmenting $CG \rightarrow I$ to infer $CG \rightarrow CGI$,
      and augmenting of $CG \rightarrow H$ to infer $CGI \rightarrow HI$,
      and then transitivity

Closure of Functional Dependencies (Cont.)

- Additional rules:
  - If $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds, then $\alpha \rightarrow \beta\gamma$ holds **(union)**
  - If $\alpha \rightarrow \beta\gamma$ holds, then $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds **(decomposition)**
  - If $\alpha \rightarrow \beta$ holds and $\gamma\beta \rightarrow \delta$ holds, then $\alpha\gamma \rightarrow \delta$ holds **(pseudotransitivity)**

  The above rules can be inferred from Armstrong's axioms.

Closure of Attribute Sets

- Given a set of attributes α, define the ***closure*** of α **under** *F* as a set of attributes that are functionally determined by α under *F*
  - Denoted by α^+

- Algorithm to compute α^+, the closure of α under *F*

result := α;
while (changes to *result*) **do**
  for each $\beta \rightarrow \gamma$ **in** *F* **do**
   begin
    if $\beta \subseteq$ *result* **then** *result* := *result* $\cup \gamma$
   end

Example of Attribute Set Closure

- $R = (A, B, C, G, H, I)$
- $F = \{A \rightarrow B$
  $A \rightarrow C$
  $CG \rightarrow H$
  $CG \rightarrow I$
  $B \rightarrow H\}$
- $(AG)^+$
  1. *result* = *AG*
  2. *result* = *ABCG* ($A \rightarrow C$ and $A \rightarrow B$)
  3. *result* = *ABCGH* ($CG \rightarrow H$ and $CG \subseteq AGBC$)
  4. *result* = *ABCGHI* ($CG \rightarrow I$ and $CG \subseteq AGBCH$)

Uses of Attribute Closure

There are several uses of the attribute closure algorithm:

- Testing for superkey:
  - To test if α is a superkey, we compute $\alpha^{+,}$ and check if α^+ contains all attributes of *R*.
- Testing functional dependencies
  - To check if a functional dependency $\alpha \rightarrow \beta$ holds (or, in other words, is in F^+), just check if $\beta \subseteq \alpha^+$.
    - That is, we compute α^+ by using attribute closure, and then check if it contains β.
    - It is a simple and cheap test, and very useful.
- **Computing closure of F (F^+)**
  - For each $\gamma \subseteq R$, we find the closure γ^+, and for each $S \subseteq \gamma^+$, we output a functional dependency $\gamma \rightarrow S$.

Example of Test for Candidate Key

- $R = (A, B, C, G, H, I)$
- $F = \{A \rightarrow B$
  $A \rightarrow C$
  $CG \rightarrow H$
  $CG \rightarrow I$
  $B \rightarrow H\}$
- Is *AG* a candidate key?
  1. Is AG a super key?
     1. Does $AG \rightarrow R$? $\Leftrightarrow$ Is $(AG)^+ \supseteq R$?
        - $(AG)^+ = ABCGHI$
  2. Is any subset of AG a superkey?
     1. Does $A \rightarrow R$? $\Leftrightarrow$ Is $(A)^+ \supseteq R$?
     2. Does $G \rightarrow R$? $\Leftrightarrow$ Is $(G)^+ \supseteq R$?

Design Goals

- Goal for a relational database design is:
  - BCNF, and
  - Lossless, and
  - Dependency preservation.
- If we cannot achieve this, we accept one of
  - Lack of dependency preservation
  - Redundancy due to use of 3NF

- Interestingly, SQL does not provide a direct way of specifying functional dependencies other than super-keys.
  - Can specify functional dependences using assertions, but they are expensive to test, and currently not supported by any of the widely used databases!
- Even if we had a dependency preserving decomposition, using SQL we would not be able to efficiently test a functional dependency whose left hand side is not a key.

Lossless Decomposition

- For the case of $R = (R_1, R_2)$, we require that for all possible relations r on schema R

$$r = \Pi_{R1}(r) \bowtie \Pi_{R2}(r)$$

- A decomposition of R into R_1 and R_2 is lossless if at least one of the following dependencies is in F^+:
  - $R_1 \cap R_2 \rightarrow R_1$
  - $R_1 \cap R_2 \rightarrow R_2$
- The above functional dependencies are a sufficient condition for lossless decomposition.
- The dependencies are a necessary condition only if all constraints are functional dependencies.

Dependency Preservation

- Let F_i be the set of dependencies F^+ that include only attributes in R_i.
  - A decomposition is **dependency preserving**, if

    $$(F_1 \cup F_2 \cup \ldots \cup F_n)^+ = F^+$$

  - If it is not, then checking updates for violation of functional dependencies may require computing joins, which is expensive.

Example

- $R = (A, B, C)$
  $F = \{ A \rightarrow B$
  $B \rightarrow C \}$
  Key = $\{A\}$
- R is not in BCNF
- Decomposition $R_1 = (A, B)$, $R_2 = (B, C)$
  - R_1 and R_2 in BCNF
  - Lossless decomposition
  - Dependency preserving

- Alternative decomposition $R_1 = (A, B)$, $R_2 = (A, C)$
  - Lossless decomposition?

    $R_1 \cap R_2 = \{A\}$ and $A \rightarrow AB$
  - Dependency preserving?

    We cannot check $B \rightarrow C$ without computing $R_1 \bowtie R_2$

First Normal Form

- Domain is **atomic** if its elements are indivisible units
  - Examples of non-atomic domains:
    - Set of names, composite attributes
    - Identification numbers like *CS101* that can be broken up into parts (department code and course id)
- A relational schema R is in **first normal form** if the domains of all attributes of R are atomic
- Non-atomic values complicate storage and encourage redundant (repeated) storage of data
  - Example
    - Set of accounts stored with each customer, and set of owners stored with each account
- We assume all relations are in first normal form

First Normal Form (Cont.)

- Atomicity is a property of how the elements of the domain are used.
  - Example
    - Strings would normally be considered indivisible
  - Suppose that students are given roll numbers which are strings of the form *CS0012* or *EE1127*
    - If the first two characters are extracted to find the department, the domain of roll numbers is not atomic.
    - Doing so is a bad idea:
      - leads to encoding of information in application program rather than in the database.

Boyce-Codd Normal Form

- A relation schema *R* is in BCNF with respect to a set *F* of functional dependencies if for all functional dependencies in F^+ of the form

$$\alpha \rightarrow \beta$$

  where $\alpha \subseteq R$ and $\beta \subseteq R$, at least one of the following holds:
  - $\alpha \rightarrow \beta$ is trivial (i.e., $\beta \subseteq \alpha$)
  - α is a superkey for *R* (i.e., $\alpha \rightarrow$ R)

- Example schema *not* in BCNF:

  instr_dept (<u>*ID*</u>*, name, salary, dept_name, building, budget*)

  - because *dept_name* $\rightarrow$ *building, budget* holds on *instr_dept,* but *dept_name* is not a superkey.

Decomposing a Schema into BCNF

- Suppose we have a schema *R*
- A non-trivial dependency $\alpha \rightarrow \beta$ causes a violation of BCNF, so we decompose *R* into:
  - $R_1 = (\alpha \cup \beta)$
  - $R_2 = (R - (\beta - \alpha))$

- In our example, *dept_name* $\rightarrow$ *building, budget*
  - α = *dept_name*
  - β = *building, budget*

  and *inst_dept* is replaced by
  - $R_1 = (\alpha \cup \beta)$ = (*dept_name, building, budget*)
  - $R_2 = (R - (\beta - \alpha))$ = (*ID, name, salary, dept_name*)

instr_dept (*ID*, *name, salary, dept_name, building, budget*)

BCNF and Dependency Preservation

- Constraints, including functional dependencies, are costly to check in practice unless they pertain to only one relation

- A decomposition is ***dependency preserving***
  - If it is sufficient to <u>test only dependencies on each individual relation</u> of the decomposition in order <u>to ensure that *all*</u> functional dependencies <u>hold</u>.

- Because it is not always possible to achieve both BCNF and dependency preservation, we consider a weaker normal form, known as *third normal form.*

Third Normal Form

- A relation schema R is in **third normal form (3NF)** if for all:

  $$\alpha \to \beta \text{ in } F^+$$

  where $\alpha \subseteq R$ and $\beta \subseteq R$, at least one of the following holds:
  - $\alpha \to \beta$ is trivial (i.e., $\beta \in \alpha$)
  - α is a superkey for R
  - Each attribute A in $\beta - \alpha$ is contained in a candidate key for R.

  (NOTE: each attribute may be in a different candidate key)

- If a relation is in BCNF, it is in 3NF
  - Since in BCNF one of the first two conditions above must hold.

- Third condition is the minimal relaxation of BCNF to ensure dependency preservation.

BCNF and Dependency Preservation

- It is not always possible to get a BCNF decomposition that is dependency preserving.
- Relation *dept_study_advisor* (*s_ID, a_ID, dept_name)*
  $F = \{ s\_ID, dept\_name \rightarrow a\_ID,$
  $a\_ID \rightarrow dept\_name \}$
  Two candidate keys = *s_ID, dept_name* and
  s_ID, a_ID
- *dept_study_advisor* is not in BCNF
- Any decomposition of *dept_study_advisor* will fail to preserve

$$s\_ID, dept\_name \rightarrow a\_ID$$

This implies that testing for $s\_ID, dept\_name \rightarrow a\_ID$ requires a join.

3NF Example

- Relation *dept_study_advisor*:
  - *dept_study_advisor* (*s_ID, a_ID, dept_name)*
    $F = \{s\_ID, dept\_name \rightarrow a\_ID,$
    $a\_ID \rightarrow dept\_name\}$
  - Two candidate keys:
    s_ID, dept_name,
    a_ID, s_ID
  - *dept_study_advisor* is in 3NF
    - $s\_ID, dept\_name \rightarrow a\_ID$
      - *s_ID, dept_name* is a superkey
    - $a\_ID \rightarrow dept\_name$
      - *a_ID* is not a superkey
      - *dept_name* is contained in a candidate key

Redundancy in 3NF

- There is some redundancy in this schema
- Example of problems due to redundancy in 3NF
  - *dept_study_advisor* (*s_ID, a_ID, dept_name)*
    F = {*s_ID, dept_name* → *a_ID,*
    a_ID → *dept_name*}

s_ID	*a_ID*	*dept_name*
Adam	*Jane*	*FI*
Bob	*Jane*	*FI*
Joe	*Jane*	*FI*
null	*Karol*	*ESF*

- repetition of information (e.g., the relationship *Jane, FI*)
  - e.g., (*a_ID, dept_name)*
- need to use *null* values (e.g., to represent the relationship *Karol, ESF* where there is no corresponding value for *s_ID*).
  - e.g., a relation (*a_ID, dept_name*) must exist if there is no other separate relation mapping instructors to departments

Second Normal Form

- A functional dependency $\alpha \rightarrow \beta$ is called **a partial dependency**
  - if there is a subset γ of α, i.e., $\gamma \subset \alpha$, such that $\gamma \rightarrow \beta$.
- We say that β is **partially dependent** on α.

- A relation *R* is in **second normal form** (2NF) if it is in 1NF and each attribute *A* in *R* meets one of the following:
  - *A* appears in a candidate key;
  - *A* <u>is not</u> partially dependent on <u>any</u> candidate key.
    - i.e., *A* is dependent on a complete candidate key, but it may be a transitive dependence.
- Every 3NF is in 2NF.

Testing for BCNF

- To check if a non-trivial dependency $\alpha \rightarrow \beta$ causes a violation of BCNF
  1. compute α^+ (the attribute closure of α), and
  2. verify that it includes all attributes of R, that is, it is a superkey of R.
- **Simplified test**: To check if a relation schema R is in BCNF, it suffices to check only the dependencies in the given set F for violation of BCNF, rather than checking all dependencies in F^+.
  - If none of the dependencies in F causes a violation of BCNF, then none of the dependencies in F^+ will cause a violation of BCNF either.
- However, **simplified test using only *F* is incorrect when testing a relation in a decomposition of R**
  - Consider $R = (A, B, C, D, E)$, with $F = \{ A \rightarrow B, BC \rightarrow D\}$
    - Decompose R into $R_1 = (\underline{A}, B)$ and $R_2 = (\underline{A}, \underline{C}, D, \underline{E})$
    - Neither of the dependencies in F contain only attributes from (A,C,D,E) so we might be misled into thinking R_2 satisfies BCNF.
    - In fact, dependency $AC \rightarrow D$ in F^+ shows R_2 is not in BCNF.

Testing Decomposition for BCNF

- To check if a relation R_i in a decomposition of R is in BCNF,
  - Either test R_i for BCNF with respect to the **restriction** of F^+ to R_i (that is, all dependences in F^+ that contain only attributes from R_i)
  - or use the original set of dependencies F that hold on R, but with the following test:
    - for every set of attributes $\alpha \subseteq R_i$, check that α^+ (the attribute closure of α) either includes no attribute of $R_i - \alpha$, or includes all attributes of R_i.

So it is a trivial FD.

So α is a superkey.

  - If the condition is violated by some $\alpha \rightarrow \beta$ in F, the dependency
    $$\alpha \rightarrow (\alpha^+ - \alpha) \cap R_i$$
    can be shown to hold on R_i, and R_i violates BCNF.
  - We use above dependency to decompose R_i

BCNF Decomposition Algorithm

result := {*R* }; -- a set of relational schemata
done := false;
compute F^+;
while (not *done)* **do**
if (there is a schema R_i in *result* that is not in BCNF)
then begin
let $\alpha \rightarrow \beta$ be a nontrivial functional dependency that
holds on R_i such that $\alpha \rightarrow R_i$ is not in F^+,
and $\alpha \cap \beta = \emptyset$;
result := (*result* – R_i) $\cup$ ($R_i - \beta$) $\cup$ (α, β);
end
else *done* := **true;**

Note: each R_i is in BCNF, and decomposition is lossless.

Example of BCNF Decomposition

- *class* (*course_id*, *title*, *dept_name*, *credits*, *sec_id*, *semester*, *year*, *building*, *room_number*, *capacity*, *time_slot_id*)
- Functional dependencies:
  - *course_id* → *title*, *dept_name*, *credits*
  - *building*, *room_number* → *capacity*
  - *course_id*, *sec_id*, *semester*, *year* → *building*, *room_number*, *time_slot_id*
- A candidate key {*course_id*, *sec_id*, *semester*, *year*}.
- BCNF Decomposition:
  - *course_id* → *title*, *dept_name*, *credits* holds
    - but *course_id* is not a superkey.
  - We replace *class* by:
    - *course*(*course_id*, *title*, *dept_name*, *credits*)
    - *class-1* (*course_id*, *sec_id*, *semester*, *year*, *building*, *room_number, capacity*, *time_slot_id*)

BCNF Decomposition (Cont.)

- *course*(*course_id*, *title*, *dept_name*, *credits*)
- *class-1* (*course_id*, *sec_id*, *semester*, *year*, *building*, *room_number, capacity*, *time_slot_id*)

- *course* is in BCNF
  - How do we know this?
- *building*, *room_number* → *capacity* holds on *class-1*
  - but {*building*, *room_number*} is not a superkey for *class-1*.
  - We replace *class-1* by:
    - *classroom* (*building*, *room_number*, *capacity*)
    - *section* (*course_id*, *sec_id*, *semester*, *year*, *building*, *room_number*, *time_slot_id*)
- *classroom* and *section* are in BCNF.

course_id → *title*, *dept_name*, *credits*
building, *room_number* → *capacity*
course_id, *sec_id*, *semester*, *year* → *building*, *room_number*, *time_slot_id*

Testing for 3NF

- Optimization
  - Need to check only dependences in *F.*
  - Need not check all dependences in F^+.
- Use attribute closure to check for each dependency $\alpha \rightarrow \beta$, if α is a superkey.
- If α is not a superkey, we have to verify whether each attribute in β-α is contained in a candidate key of *R*
  - This test is rather more expensive, since it involves finding candidate keys.
  - Testing for 3NF has been shown to be NP-hard.
  - Interestingly, decomposition into third normal form (described shortly) can be done in polynomial time.