Constraints: Functional Dependencies
Fall 2017

School of Computer Science
University of Waterloo

Databases CS348
Schema Design

When we get a relational schema,

⇒ **how do we know if its any good?**
⇒ **what to watch for?**

- what are the allowed instances of the schema?
- does the structure capture the data?
  ⇒ too hard to query?
  ⇒ too hard to **update**?
  ⇒ redundant information all over the place?
Change Anomalies

Assume we are given the E-R diagram
Change Anomalies (cont.)

Supplied_Items

<table>
<thead>
<tr>
<th>Sno</th>
<th>Sname</th>
<th>City</th>
<th>Ino</th>
<th>Iname</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>Magna</td>
<td>Ajax</td>
<td>I1</td>
<td>Bolt</td>
<td>0.50</td>
</tr>
<tr>
<td>S1</td>
<td>Magna</td>
<td>Ajax</td>
<td>I2</td>
<td>Nut</td>
<td>0.25</td>
</tr>
<tr>
<td>S1</td>
<td>Magna</td>
<td>Ajax</td>
<td>I3</td>
<td>Screw</td>
<td>0.30</td>
</tr>
<tr>
<td>S2</td>
<td>Budd</td>
<td>Hull</td>
<td>I3</td>
<td>Screw</td>
<td>0.40</td>
</tr>
</tbody>
</table>

Problems:

1. Update problems (Changing name of supplier)
2. Insert problems (New item w/o supplier)
3. Delete problems (Budd no longer supplies screws)
4. Likely increase in space requirements
Change Anomalies (cont.)

Compare to

<table>
<thead>
<tr>
<th>Supplier</th>
<th>Sno</th>
<th>Sname</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>Magna</td>
<td>Ajax</td>
<td></td>
</tr>
<tr>
<td>S2</td>
<td>Budd</td>
<td>Hull</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Item</th>
<th>Ino</th>
<th>Iname</th>
</tr>
</thead>
<tbody>
<tr>
<td>I1</td>
<td>1</td>
<td>Bolt</td>
</tr>
<tr>
<td>I2</td>
<td>2</td>
<td>Nut</td>
</tr>
<tr>
<td>I3</td>
<td>3</td>
<td>Screw</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Supplies</th>
<th>Sno</th>
<th>Ino</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>S1</td>
<td>I1</td>
<td>0.50</td>
</tr>
<tr>
<td></td>
<td>S1</td>
<td>I2</td>
<td>0.25</td>
</tr>
<tr>
<td></td>
<td>S1</td>
<td>I3</td>
<td>0.30</td>
</tr>
<tr>
<td></td>
<td>S2</td>
<td>I3</td>
<td>0.40</td>
</tr>
</tbody>
</table>

Decomposition seems to be better…
But other extreme is also undesirable
⇒ information about relationships can be lost

... so how do we know how much can we decompose?
How to Find and Fix Anomalies?

**Detection:** How do we know an anomaly exists?

(certain families) of **Integrity Constraints** postulate regularities in schema instances that lead to anomalies.

**Repair** How can we fix it?

Certain **Schema Decompositions** avoid the anomalies while retaining *all information* in the instances.
Integrity Constraints

**Idea:** allow only **well-behaved** instances of the schema

⇒ the relational structure (= selection of relations)

is often not sufficient to capture all of these.

- restrict values of an attribute
- describe dependencies between attributes
  ⇒ in a single relation (bad)
  ⇒ between relations (good)
- postulate the existence of values in the database
- . . .

Dependencies between attributes in a single relation lead to improvements in schema design.
Functional Dependencies (FDs)

**Idea:** to express the fact that in a relation schema (values of) a set of attributes uniquely determine (values of) another set of attributes.

**Definition:** Let $R$ be a relation schema, and $X, Y \subseteq R$ sets of attributes. The functional dependency $X \rightarrow Y$ is the formula

\[
\forall v_1, \ldots, v_k, w_1, \ldots, w_k. R(v_1, \ldots, v_k) \land R(w_1, \ldots, w_k) \land
\left(\bigwedge_{j \in X} v_j = w_j\right) \rightarrow \left(\bigwedge_{i \in Y} v_i = w_i\right)
\]

We say that (the set of attributes) $X$ functionally determines $Y$ (in $R$).
Examples of Functional Dependencies

Consider the following relation schema:

\[
\text{EmpProj} = \begin{array}{|c|c|c|c|c|c|c|}
\hline
\text{SIN} & \text{PNum} & \text{Hours} & \text{EName} & \text{PName} & \text{PLoc} & \text{Allowance} \\
\hline
\end{array}
\]

- SIN determines employee name
  \[\text{SIN} \rightarrow \text{EName}\]
- project number determines project name and location
  \[\text{PNum} \rightarrow \text{PName}, \text{PLoc}\]
- allowances are always the same for the same number of hours at the same location
  \[\text{PLoc}, \text{Hours} \rightarrow \text{Allowance}\]
Implication for FDs

How do we know what additional FDs hold in a schema?

A set $F$ logically implies a FD $X \rightarrow Y$ if $X \rightarrow Y$ holds in all instances of $R$ that satisfy $F$.

The closure of $F^+$ of $F$ is the set of all functional dependencies that are logically implied by $F$.

Clearly: $F \subseteq F^+$, but what else is in $F^+$?

For Example:

$F = \{A \rightarrow B, B \rightarrow C\}$ then $F^+ = \{A \rightarrow B, B \rightarrow C, A \rightarrow C\}$
Reasoning About FDs

Logical implications can be derived by using inference rules called Armstrong’s axioms

- (reflexivity) $Y \subseteq X \Rightarrow X \rightarrow Y$
- (augmentation) $X \rightarrow Y \Rightarrow XZ \rightarrow YZ$
- (transitivity) $X \rightarrow Y, Y \rightarrow Z \Rightarrow X \rightarrow Z$

The axioms are

- sound (anything derived from $F$ is in $F^+$)
- complete (anything in $F^+$ can be derived)

Additional rules can be derived

- (union) $X \rightarrow Y, X \rightarrow X \Rightarrow X \rightarrow YZ$
- (decomposition) $X \rightarrow YZ \Rightarrow X \rightarrow Y$
Reasoning (example)

Example: \( F = \{ \text{SIN, PNum} \rightarrow \text{Hours} \)
\( \text{SIN} \rightarrow \text{EName} \)
\( \text{PNum} \rightarrow \text{PName, PLoc} \)
\( \text{PLoc, Hours} \rightarrow \text{Allowance} \} \)

A derivation of \( \text{SIN, PNum} \rightarrow \text{Allowance} \):
1. \( \text{SIN, PNum} \rightarrow \text{Hours} \) (\( \in F \))
2. \( \text{PNum} \rightarrow \text{PName, PLoc} \) (\( \in F \))
3. \( \text{PLoc, Hours} \rightarrow \text{Allowance} \) (\( \in F \))
4. \( \text{SIN, PNum} \rightarrow \text{PNum} \) (reflexivity)
5. \( \text{SIN, PNum} \rightarrow \text{PName, PLoc} \) (transitivity, 4 and 2)
6. \( \text{SIN, PNum} \rightarrow \text{PLoc} \) (decomposition, 5)
7. \( \text{SIN, PNum} \rightarrow \text{PLoc, Hours} \) (union, 6, 1)
8. \( \text{SIN, PNum} \rightarrow \text{Allowance} \) (transitivity, 7 and 3)
Keys: formal definition

Definition:
- $K \subseteq R$ is a **superkey** for relation schema $R$ if dependency $K \rightarrow R$ holds on $R$.

- $K \subseteq R$ is a **candidate key** for relation schema $R$ if $K$ is a superkey and no subset of $K$ is a superkey.

**Primary Key** = a candidate key chosen by the DBA.
Efficient Reasoning

How to figure out if an FD is implied by $F$ quickly?

⇒ a mechanical and more efficient way of using Armstrong’s axioms:

```
function ComputeX⁺(X, F)
begin
    X⁺ := X;
    while true do
        if there exists $(Y \rightarrow Z) \in F$ such that
            (1) $Y \subseteq X⁺$, and
            (2) $Z \not\subseteq X⁺$
        then $X⁺ := X⁺ \cup Z$
        else exit;
    return $X⁺$;
end
```
Let $R$ be a relational schema and $F$ a set of functional dependencies on $R$. Then

**Theorem:** $X$ is a superkey of $R$ if and only if

$$\text{Compute}X^+(X, F) = R$$

**Theorem:** $X \rightarrow Y \in F^+$ if and only if

$$Y \subseteq \text{Compute}X^+(X, F)$$
Computing a Decomposition

Let $R$ be a relation schema (= set of attributes). The collection \( \{R_1, \ldots, R_n\} \) of relation schemas is a decomposition of $R$ if

\[
R = R_1 \cup R_2 \cup \cdots \cup R_n
\]

A good decomposition does not

- lose information
- complicate checking of constraints
- contain anomalies (or at least contains fewer anomalies)
Lossless-Join Decompositions

We should be able to construct the instance of the original table from the instances of the tables in the decomposition

**Example:** Consider replacing

<table>
<thead>
<tr>
<th>Marks</th>
<th>Student</th>
<th>Assignment</th>
<th>Group</th>
<th>Mark</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Ann</td>
<td>A1</td>
<td>G1</td>
<td>80</td>
</tr>
<tr>
<td></td>
<td>Ann</td>
<td>A2</td>
<td>G3</td>
<td>60</td>
</tr>
<tr>
<td></td>
<td>Bob</td>
<td>A1</td>
<td>G2</td>
<td>60</td>
</tr>
</tbody>
</table>

by decomposing to two tables

<table>
<thead>
<tr>
<th>SGM</th>
<th>Student</th>
<th>Group</th>
<th>Mark</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Ann</td>
<td>G1</td>
<td>80</td>
</tr>
<tr>
<td></td>
<td>Ann</td>
<td>G3</td>
<td>60</td>
</tr>
<tr>
<td></td>
<td>Bob</td>
<td>G2</td>
<td>60</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>AM</th>
<th>Assignment</th>
<th>Mark</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A1</td>
<td>80</td>
</tr>
<tr>
<td></td>
<td>A2</td>
<td>60</td>
</tr>
<tr>
<td></td>
<td>A1</td>
<td>60</td>
</tr>
</tbody>
</table>
But computing the natural join of SGM and AM produces

<table>
<thead>
<tr>
<th>Student</th>
<th>Assignment</th>
<th>Group</th>
<th>Mark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ann</td>
<td>A1</td>
<td>G1</td>
<td>80</td>
</tr>
<tr>
<td>Ann</td>
<td>A2</td>
<td>G3</td>
<td>60</td>
</tr>
<tr>
<td>Ann</td>
<td>A1</td>
<td>G3</td>
<td>60 !</td>
</tr>
<tr>
<td>Bob</td>
<td>A2</td>
<td>G2</td>
<td>60 !</td>
</tr>
<tr>
<td>Bob</td>
<td>A1</td>
<td>G2</td>
<td>60</td>
</tr>
</tbody>
</table>

... and we get extra data (spurious tuples) and would therefore lose information if we were to replace Marks by SGM and AM.

If re-joining SGM and AM would always produce exactly the tuples in Marks, then we call SGM and AM a lossless-join decomposition.
A decomposition $\{R_1, R_2\}$ of $R$ is lossless if and only if the common attributes of $R_1$ and $R_2$ form a superkey for either schema, that is

$$R_1 \cap R_2 \rightarrow R_1 \text{ or } R_1 \cap R_2 \rightarrow R_2$$

**Example:** In the previous example we had

$$R = \{\text{Student, Assignment, Group, Mark}\},$$

$$F = \{(\text{Student, Assignment} \rightarrow \text{Group, Mark})\},$$

$$R_1 = \{\text{Student, Group, Mark}\},$$

$$R_2 = \{\text{Assignment, Mark}\}$$

$\Rightarrow$ decomposition $\{R_1, R_2\}$ is lossy because

$$R_1 \cap R_2 ( = \{M\}) \text{ is not a superkey of either SGM or AM}$$
Dependency Preservation

How do we test/enforce constraints on the decomposed schema?

**Example:** A table for a company database could be

<table>
<thead>
<tr>
<th>R</th>
<th>Proj</th>
<th>Dept</th>
<th>Div</th>
</tr>
</thead>
<tbody>
<tr>
<td>FD1: Proj $\rightarrow$ Dept,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FD2: Dept $\rightarrow$ Div, and</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FD3: Proj $\rightarrow$ Div</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

and two decompositions

$D_1 = \{R1[Proj, Dept], R2[Dept, Div]\}$

$D_2 = \{R1[Proj, Dept], R3[Proj, Div]\}$

Both are lossless. (Why?)
Dependency Preservation (cont.)

Which decomposition is better?

- Decomposition $D_1$ lets us test FD1 on table R1 and FD2 on table R2; if they are both satisfied, FD3 is automatically satisfied.

- In decomposition $D_2$ we can test FD1 on table R1 and FD3 on table R3. Dependency FD2 is an **interrelational constraint**: testing it requires joining tables R1 and R3.

$\Rightarrow D_1$ is better!

A decomposition $D = \{R_1, \ldots, R_n\}$ of $R$ is **dependency preserving** if there is an equivalent set $F'$ of functional dependencies, none of which is interrelational in $D$. 
Avoiding Anomalies

What is a “good” relational database schema?

Rule of thumb: Independent facts in separate tables:

“Each relation schema should consist of a **primary key** and a **set of mutually independent attributes**”

⇒ achieved by transformation of a schema to a **normal form**

**Goals:**

- Intuitive and straightforward changes
- Anomaly-free/Nonredundant representation of data

**We discuss:**

- Boyce-Codd Normal Form (BCNF)
- Third Normal Form (3NF)

...both based on the notion of **functional dependency**
Boyce-Codd Normal Form (BCNF)

Schema $R$ is in **BCNF** (w.r.t. $F$) if and only if whenever $(X \rightarrow Y) \in F^+$ and $XY \subseteq R$, then either

- $(X \rightarrow Y)$ is trivial (i.e., $Y \subseteq X$), or
- $X$ is a superkey of $R$

A database schema $\{R_1, \ldots, R_n\}$ is in BCNF if each relation schema $R_i$ is in BCNF.

Formalization of the goal that independent relationships are stored in separate tables.
BCNF (cont.)

Why does BCNF avoid redundancy?

For the schema \textit{Supplied\_Items} we had a FD:

\[ \text{Sno} \rightarrow \text{Sname, City} \]

Therefore: supplier name “Magna” and city “Ajax” must be repeated for each item supplied by supplier S1.

Assume the above FD holds over a schema \( R \) that is in BCNF. Then:

- Sno is a superkey for \( R \)
- each Sno value appears on one row only
- no need to repeat Sname and City values
function $ComputeBCNF(R, F)$
begin
    $Result := \{R\};$
    while some $R_i \in Result$ and $(X \rightarrow Y) \in F^+$ violate the BCNF condition do begin
        Replace $R_i$ by $R_i - (Y - X);$
        Add $\{X, Y\}$ to $Result;$
    end;
    return $Result;$
end

Lossless-Join BCNF Decomposition

- No *efficient* procedure to do this exists.
- Results depend on sequence of FDs used to decompose the relations.
- It is possible that no lossless join dependency preserving BCNF decomposition exists:

  \[ R = \{A, B, C\} \text{ and } F = \{AB \rightarrow C, C \rightarrow B\}. \]
Third Normal Form (3NF)

Schema $R$ is in **3NF** (w.r.t. $F$) if and only if whenever $(X \rightarrow Y) \in F^+$ and $XY \subseteq R$, then either
- $(X \rightarrow Y)$ is trivial, or
- $X$ is a superkey of $R$, or
- each attribute of $Y$ contained in a candidate key of $R$

A schema $\{R_1, \ldots, R_n\}$ is in 3NF if each relation schema $R_i$ is in 3NF.

- 3NF is looser than BCNF
  - ⇒ allows more redundancy
  - ⇒ $R = \{A, B, C\}$ and $F = \{AB \rightarrow C, C \rightarrow B\}$.
- lossless-join, dependency-preserving decomposition into 3NF relation schemas always exists.
Minimal Cover

**Definition:** Two sets of dependencies $F$ and $G$ are **equivalent** iff $F^+ = G^+$.

There are different sets of functional dependencies that have the same logical implications. Simple sets are desirable.

**Definition:** A set of dependencies $G$ is **minimal** if

1. every right-hand side of an dependency in $F$ is a single attribute.
2. for no $X \rightarrow A$ is the set $F - \{X \rightarrow A\}$ equivalent to $F$.
3. for no $X \rightarrow A$ and $Z$ a proper subset of $X$ is the set $F - \{X \rightarrow A\} \cup \{Z \rightarrow A\}$ equivalent to $F$.

**Theorem:** For every set of dependencies $F$ there is an equivalent minimal set of dependencies (**minimal cover**).
Finding Minimal Covers

A minimal cover for $F$ can be computed in four steps. Note that each step must be repeated until it no longer succeeds in updating $F$.

**Step 1.**
Replace $X \rightarrow YZ$ with the pair $X \rightarrow Y$ and $X \rightarrow Z$.

**Step 2.**
Remove $X \rightarrow A$ from $F$ if $A \in ComputeX^+(X, F - \{X \rightarrow A\})$.

**Step 3.**
Remove $A$ from the left-hand-side of $X \rightarrow B$ in $F$ if $B$ is in $ComputeX^+(X - \{A\}, F)$.

**Step 4.**
Replace $X \rightarrow Y$ and $X \rightarrow Z$ in $F$ by $X \rightarrow YZ$. 
A lossless-join 3NF decomposition that is dependency preserving can be efficiently computed

```
function Compute3NF(R, F)
begin
    Result := ∅;
    F′ := a minimal cover for F;
    for each (X → Y) ∈ F′ do
        Result := Result ∪ {XY};
    if there is no Rᵢ ∈ Result such that
        Rᵢ contains a candidate key for R then begin
            compute a candidate key K for R;
            Result := Result ∪ {K};
        end;
    return Result;
end
```
Summary

- functional dependencies provide clues towards elimination of (some) *redundancies* in a relational schema.
- Goals: to decompose relational schemas in such a way that the decomposition is
  
  (1) lossless-join
  (2) dependency preserving
  (3) BCNF (and if we fail here, at least 3NF)