DATABASE DESIGN

Functional Dependencies and Redundancy

University of Waterloo
# List of Slides

1. Schema Design
2. Change Anomalies
3. Change Anomalies (cont.)
4. Change Anomalies (cont.)
5. Change Anomalies (cont.)
6. Change Anomalies (cont.)
7. Integrity Constraints
8. Functional Dependencies (FDs)
9. Examples of Functional Dependencies
10. Implication for FDs
11. Reasoning About FDs
12. Reasoning (example)
13. Keys: formal definition
14. Efficient Reasoning
15. Efficient Reasoning (cont.)
16. Good Database Design
17. Normal Forms and Decomposition
18. Boyce-Codd Normal Form (BCNF)
19. Computing a Normal Form
20. Lossless-Join Decompositions
21. Lossless-Join Decompositions (cont.)
22. Lossless-Join Decompositions (cont.)
23. Dependency Preservation
24. Dependency Preservation (cont.)
25. Lossless-Join BCNF Decomposition
26. Lossless-Join BCNF Decomposition
27. Third Normal Form (3NF)
28. Third Normal Form (3NF)
29. Minimal Covers
30. Finding Minimal Covers
31. Computing a 3NF Decomposition
32. Summary
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

```
Ino  Sno        Sname
     Iname     Price
```

```
Supplied_Items
     City
```
Change Anomalies (cont.)

This maps to

<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 (e.g. changing name of supplier)
2. Insert problems (e.g. add a new item)
3. Delete problems (Budd no longer supplies screws)
4. Likely increase in space requirements
Change Anomalies (cont.)

Now compare to

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

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

<table>
<thead>
<tr>
<th>Sno</th>
<th>Ino</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>I1</td>
<td>0.50</td>
</tr>
<tr>
<td>S1</td>
<td>I2</td>
<td>0.25</td>
</tr>
<tr>
<td>S1</td>
<td>I3</td>
<td>0.30</td>
</tr>
<tr>
<td>S2</td>
<td>I3</td>
<td>0.40</td>
</tr>
</tbody>
</table>
Change Anomalies (cont.)

But other extreme is also undesirable (information about relationships is lost)

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

<table>
<thead>
<tr>
<th>Inums</th>
<th>Inames</th>
<th>Prices</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inum</td>
<td>Iname</td>
<td>Price</td>
</tr>
<tr>
<td>I1</td>
<td>Bolt</td>
<td>0.50</td>
</tr>
<tr>
<td>I2</td>
<td>Nut</td>
<td>0.25</td>
</tr>
<tr>
<td>I3</td>
<td>Screw</td>
<td>0.30</td>
</tr>
</tbody>
</table>
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.

Notation: projection operation on tuples:

\[ t[A_1, \ldots, A_k] = (t.A_1, \ldots, t.A_k) \]

Definition: Let \( R \) be a relation schema, and \( X, Y \subseteq R \) sets of attributes. The functional dependency

\[ X \rightarrow Y \]

holds on \( R \) if whenever an instance of \( R \) contains two tuples \( t \) and \( u \) such that \( t[X] = u[X] \) then it is also true that \( t[Y] = u[Y] \).

We say that \( X \) functionally determines \( Y \) (in \( R \)).
Examples of Functional Dependencies

Consider the following relation schema:

EmpProj

<table>
<thead>
<tr>
<th>SIN</th>
<th>PNum</th>
<th>Hours</th>
<th>EName</th>
<th>PName</th>
<th>PLoc</th>
<th>Allowance</th>
</tr>
</thead>
</table>

- 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
  
  \[ \text{PLoc, Hours} \rightarrow \text{Allowance} \]
Implication for FDs

How do we know what additional FDs hold in a schema?
⇒ does “PNum, Hours → Allowance” hold?

Let $F$ denote a set of functional dependencies over $R$. The closure of $F$, denoted by $F^+$, is the set of all functional dependencies that are satisfied by every relation instance that satisfies $F$.

$$ R |\rightarrow F \iff R |\rightarrow F^+ $$

= logical implications of $F$ (for all instances of $R$)

Clearly: $F \subseteq F^+$
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) \( X \rightarrowYZ \Rightarrow X \rightarrow Y \)
Reasoning (example)

Example: Let $F$ consist of

- $\text{SIN, PNum } \rightarrow \text{ Hours}$
- $\text{SIN } \rightarrow \text{ EName}$
- $\text{PNum } \rightarrow \text{ PName, PLoc}$
- $\text{PLoc, Hours } \rightarrow \text{ Allowance}$

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 more efficient way of using Armstrong’s axioms:

```plaintext
function Compute$X^+(X, F)$
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
```
Efficient Reasoning (cont.)

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

\[
Y \subseteq \text{Compute} X^+(X, F)
\]
Good Database Design

What is a “good” relational database schema?

Rule of thumb: Independent facts in separate tables

or: Each relation schema should consist of a primary key and a set of mutually independent attributes
Normal Forms and Decomposition

Goals:

- Intuitive and straightforward changes
- Non-redundant storage 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)

Let $R$ be a relation schema and $F$ a set of functional dependencies.

Schema $R$ is in BCNF 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.
Computing a Normal Form

What to do if a given relational schema is not in BCNF?

Strategy: identify undesirable dependencies, and decompose the schema

Definition:
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
Lossless-Join Decompositions

We should be able to construct the original table from its decomposition

Example: Consider replacing

<table>
<thead>
<tr>
<th>Marks</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Student</td>
<td>Assignment</td>
<td>Group</td>
<td>Mark</td>
</tr>
<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>Bob</td>
<td>A1</td>
<td>G2</td>
<td>60</td>
</tr>
</tbody>
</table>

by decomposing (i.e. projecting) into two tables

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

<table>
<thead>
<tr>
<th>AM</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Assignment</td>
<td>Mark</td>
<td></td>
</tr>
<tr>
<td>A1</td>
<td>80</td>
<td></td>
</tr>
<tr>
<td>A2</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td>A1</td>
<td>60</td>
<td></td>
</tr>
</tbody>
</table>
Lossless-Join Decompositions (cont.)

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>

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

If converse is true, if re-joining SGM and AM would **always** produce exactly the tuples in Marks, then we call SGM and AM a **lossless-join decomposition**.
Lossless-Join Decompositions (cont.)

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

\[
R_1 \cap R_2 \rightarrow R_1 \quad \text{or} \quad R_1 \cap R_2 \rightarrow R_2
\]

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

\[
R = \{ \text{Student, Assignment, Group, Mark} \}, \\
F = \{ (\text{Student, Assignment} \rightarrow \text{Group, Mark}) \},
\]

\[
R_1 = \{ \text{Student, Group, Mark} \}, \\
R_2 = \{ \text{Assignment, Mark} \}
\]

Decomposition \( \{ R_1, R_2 \} \) is lossy because \( R_1 \cap R_2 (= \{ M \}) \) is not a superkey of either SGM or AM
Dependency Preservation

Goal: efficient testing of constraints on the decomposed schema

Example: A table for a company database could be

\[
\begin{array}{ccc}
\text{Proj} & \text{Dept} & \text{Div} \\
\end{array}
\]

with functional dependencies

FD1: Proj → Dept,
FD2: Dept → Div, and
FD3: Proj → Div

Consider two decompositions

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

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

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

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 inter-relational constraint: testing it requires joining tables R1 and R3.

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 inter-relational in $D$. 
Lossless-Join BCNF Decomposition

function $\text{ComputeBCNF}(R, F)$

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

- Results depend on sequence of FDs used to decompose the relations.

- It is possible that no dependency preserving BCNF decomposition exists:

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

Let $R$ be a relation schema and $F$ a set of functional dependencies.

Schema $R$ is in 3NF 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 database schema \( \{ R_1, \ldots, R_n \} \) is in 3NF if each relation schema $R_i$ is in 3NF
Third Normal Form (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 Covers

**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
$$A \in \text{Compute} X^+(X, F - \{X \rightarrow A\}).$$

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

**Step 4.**
Replace $X \rightarrow Y$ and $X \rightarrow Z$ in $F$ by $X \rightarrow YZ$. 
Computing a 3NF Decomposition

A lossless-join 3NF decomposition that is dependency preserving can be efficiently computed

\begin{verbatim}
function Compute3NF(R, F)
begin
Result := \emptyset;
F' := a minimal cover for F;
for each (X \rightarrow Y) \in F' do
    Result := Result \cup \{XY\};
if there is no R_i \in Result such that
    R_i contains a candidate key for R then begin
    compute a candidate key K for R;
    Result := Result \cup \{K\};
end;
return Result;
end
\end{verbatim}
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)