DATABASE DESIGN

Functional Dependencies and Redundancy

University of Waterloo

List of Slides

Schema Design

When we get a relational schema,

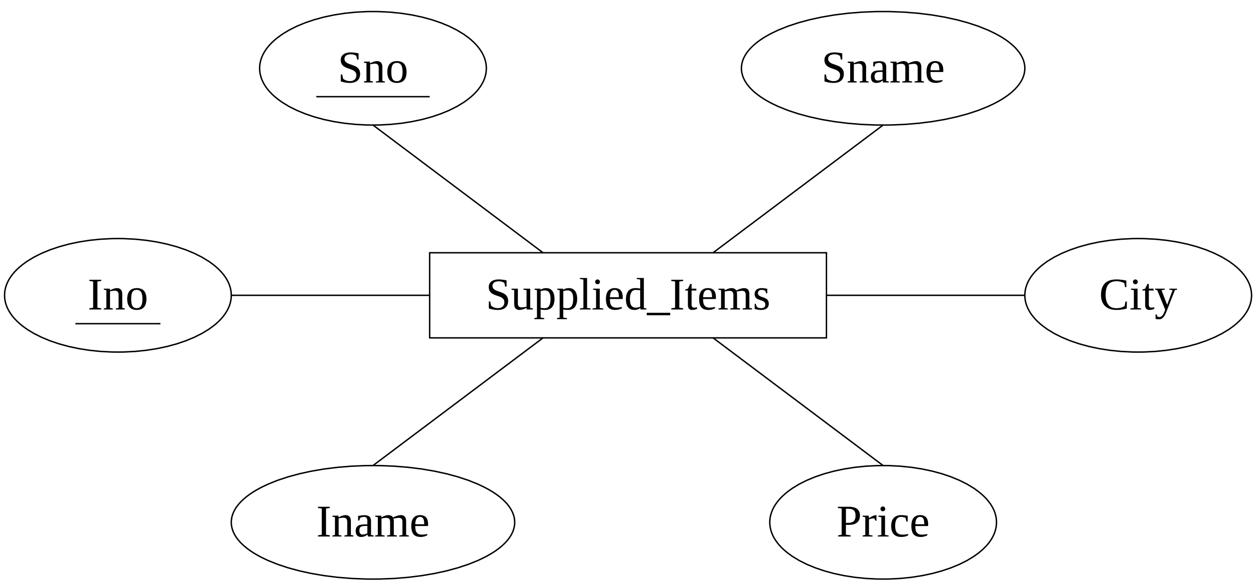
$\Rightarrow$ **how do we know if its any good**?

$\Rightarrow$ **what to watch for**?

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

Change Anomalies

Assume we are given the E-R diagram



Change Anomalies (cont.)

This maps to

Supplied_Items

Sno	Sname	City	Ino	Iname	Price
S1	Magna	Ajax	I1	Bolt	0.50
S1	Magna	Ajax	I2	Nut	0.25
S1	Magna	Ajax	I3	Screw	0.30
S2	Budd	Hull	I3	Screw	0.40

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

Supplier

Sno	Sname	City
S1	Magna	Ajax
S2	Budd	Hull

Item

Ino	Iname
I1	Bolt
I2	Nut
I3	Screw

Supplies

Sno	Ino	Price
S1	I1	0.50
S1	I2	0.25
S1	I3	0.30
S2	I3	0.40

Change Anomalies (cont.)

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

Snos

Sno
S1
S2

Snames

Sname
Magna
Budd

Cities

City
Ajax
Hull

Inums

Inum
I1
I2
I3

Inames

Iname
Bolt
Nut
Screw

Prices

Price
0.50
0.25
0.30
0.40

Integrity Constraints

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

$\Rightarrow$ the relational structure (= selection of relations)
is often not sufficient to capture all of these.

- restrict values of an attribute
- describe dependencies between attributes

  $\Rightarrow$ in a single relation (bad)
  $\Rightarrow$ 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**

$$X \to Y$$

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

<u>SIN</u>	<u>PNum</u>	Hours	EName	PName	PLoc	Allowance

- SIN determines employee name

  SIN $\rightarrow$ EName

- project number determines project name and location

  PNum $\rightarrow$ PName, PLoc

- allowances are always the same for the same number of hours at the same location

  PLoc, Hours $\rightarrow$ Allowance

Implication for FDs

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

$\Rightarrow$ does "PNum, Hours $\rightarrow$ 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.

$$R \models F \iff R \models F^+$$

= *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) $Y \subseteq X \Rightarrow X \to Y$
- (augmentation) $X \to Y \Rightarrow XZ \to YZ$
- (transitivity) $X \to Y$, $Y \to Z \Rightarrow X \to Z$

The axioms are

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

Additional rules can be derived

- (union) $X \to Y$, $X \to X \Rightarrow X \to YZ$
- (decomposition) $X \to YZ \Rightarrow X \to Y$

Reasoning (example)

Example: Let F consist of

SIN, PNum $\rightarrow$ Hours
SIN $\rightarrow$ EName
PNum $\rightarrow$ PName, PLoc
PLoc, Hours $\rightarrow$ Allowance

A derivation of: SIN, PNum $\rightarrow$ Allowance

1. SIN, PNum $\rightarrow$ Hours ($\in F$)
2. PNum $\rightarrow$ PName, PLoc ($\in F$)
3. PLoc, Hours $\rightarrow$ Allowance ($\in F$)
4. SIN, PNum $\rightarrow$ PNum (reflexivity)
5. SIN, PNum $\rightarrow$ PName, PLoc (transitivity, 4 and 2)
6. SIN, PNum $\rightarrow$ PLoc (decomposition, 5)
7. SIN, PNum $\rightarrow$ PLoc, Hours (union, 6, 1)
8. SIN, PNum $\rightarrow$ Allowance (transitivity, 7 and 3)

Keys: formal definition

Definition:

- $K \subseteq R$ is a **superkey** for relation schema R if dependency $K \to R$ 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:

function $ComputeX^+(X, F)$
begin
  X^+ := X;
  while true do
    if there exists $(Y \to Z) \in F$ 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

$$ComputeX^+(X, F) = R$$

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

$$Y \subseteq ComputeX^+(X, F)$$

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

Marks

Student	Assignment	Group	Mark
Ann	A1	G1	80
Ann	A2	G3	60
Bob	A1	G2	60

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

SGM

Student	Group	Mark
Ann	G1	80
Ann	G3	60
Bob	G2	60

AM

Assignment	Mark
A1	80
A2	60
A1	60

Lossless-Join Decompositions (cont.)

But computing the natural join of SGM and AM produces

Student	Assignment	Group	Mark
Ann	A1	G1	80
Ann	A2	G3	60
Ann	A1	G3	60 !
Bob	A2	G2	60 !
Bob	A1	G2	60

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

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

Example: In the previous example we had

R = $\{$Student, Assignment, Group, Mark$\}$,
F = $\{$(Student, Assignment $\to$ Group, Mark)$\}$,

R_1 = $\{$Student, Group, Mark$\}$,
R_2 = $\{$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

R

Proj	Dept	Div

with functional dependencies

FD1: Proj $\rightarrow$ Dept,
FD2: Dept $\rightarrow$ Div, and
FD3: Proj $\rightarrow$ Div

Consider 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.)

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 $ComputeBCNF(R, F)$

begin

Result := $\{R\}$;

while some $R_i \in$ Result and $(X \to Y) \in F^+$

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 $(X \to Y) \in F^+$ and $XY \subseteq R$, then either

- $(X \to Y)$ 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

  $\Rightarrow$ allows more redundancy

  $\Rightarrow R = \{$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 $X \to A$ is the set $F - \{X \to A\}$ equivalent to F.
3. for no $X \to A$ and Z a proper subset of X is the set $F - \{X \to A\} \cup \{Z \to A\}$ 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 $X \to YZ$ with the pair $X \to Y$ and $X \to Z$.

Step 2.
Remove $X \to A$ from F if

$$A \in ComputeX^+(X, F - \{X \to A\}).$$

Step 3.
Remove A from the left-hand-side of $X \to B$ in F if

$$B \text{ is in } ComputeX^+(X - \{A\}, F).$$

Step 4.
Replace $X \to Y$ and $X \to Z$ in F by $X \to YZ$.

Computing a 3NF Decomposition

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

function $Compute3NF(R, F)$
begin
    Result := $\emptyset$;
    F' := a minimal cover for F;
    for each $(X \to Y) \in F'$ **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

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)